TNet: A Model-Constrained Tikhonov Network Approach for Inverse Problems*

Hai V. Nguyen† and Tan Bui-Thanh‡

Abstract. Deep Learning (DL), in particular deep neural networks (DNN), by default is purely data-driven and in general does not require physics. This is the strength of DL but also one of its key limitations when applied to science and engineering problems in which underlying physical properties—such as stability, conservation, and positivity—and accuracy are required. DL methods in their original forms are often not capable of respecting the underlying mathematical models or achieving desired accuracy even in big-data regimes. On the other hand, many data-driven science and engineering problems, such as inverse problems, typically have limited experimental or observational data, and DL would overfit the data in this case. Leveraging information encoded in the underlying mathematical models, we argue, not only compensates missing information in low data regimes but also provides opportunities to equip DL methods with the underlying physics, and hence promoting better generalization. This paper develops a model-constrained deep learning approach and its variant `TNet`—a Tikhonov neural network—that are capable of learning not only information hidden in the training data but also in the underlying mathematical models to solve inverse problems governed by partial differential equations in low data regimes. We provide the constructions and some theoretical results for the proposed approaches for both linear and nonlinear inverse problems. Since TNet is designed to learn inverse solution with Tikhonov regularization, it is interpretable: in fact it recovers Tikhonov solutions for linear cases while potentially approximating Tikhonov solutions for nonlinear inverse problems. We also prove that data randomization can enhance not only the smoothness of the networks but also their generalizations. Comprehensive numerical results confirm the theoretical findings and show that with even as little as 1 training data sample for 1D deconvolution, 5 for inverse 2D heat conductivity problem, 100 for inverse initial conditions for time-dependent 2D Burgers' equation, and 50 for inverse initial conditions for 2D Navier-Stokes equations, `TNet` solutions can be as accurate as Tikhonov solutions while being several orders of magnitude faster. This is possible owing to the model-constrained term, replications, and randomization.

Key words. Inverse problem, randomization, model-constrained, deep learning, deep neural network, partial differential equations.

1. Introduction.

Inverse problems are pervasive in scientific discovery and decision-making for complex, natural, engineered, and societal systems. They are perhaps the most popular mathematical approaches for *enabling predictive scientific simulations* that *integrate observational/experimental data, simulations and/or models* [44, 27, 60]. Many engineering and science systems are governed by parametrized partial differential equations (PDE). Computational PDE-constrained inverse problems face not only the ill-posed nature—namely, nonexistence, non-uniqueness, and instability of inverse solutions—but also the computational expense of solving the underlying PDEs. Computational inverse methods typically require the PDEs to be solved at many realizations of parameter and the cost is an (possibly ex-

*The first version of work was an Oden Institute technical report 21-09, May 11, 2021, and then presented at the Uncertainty Quantification and Probabilistic Modeling Technical Thrust Area, USNCCM, May 17, 2021.

†Department of Aerospace Engineering and Engineering Mechanics, the University of Texas at Austin, Texas (hainguyen@utexas.edu)

‡Department of Aerospace Engineering and Engineering Mechanics, The Oden Institute for Computational Engineering and Sciences, the University of Texas at Austin, Texas (tanbui@oden.utexas.edu, https://users.oden.utexas.edu/~tanbui/).

ponentially) increasing function of the parameter dimension. The fast growth of this cost is typically associated with the curse of dimensionality. Inverse problems for practical complex systems [3, 44, 30, 12, 31] however possess this high dimensional parameter space challenge. Thus, mitigating the cost of repeatedly solving the underlying PDE has been of paramount importance in computational PDE-constrained inverse problems.

The field of Machine Learning (ML) typically refers to computational and statistical methods for the automated detection of meaningful patterns in data [7, 55, 40]. While Deep Learning (DL) [21], a subset of machine learning, has proved to be state-of-the-art methods in many fields of computer sciences such as computer vision, speech recognition, natural language processing, etc, and its presence in the scientific computing community is, however, mostly limited to off-the-shelf applications of deep learning. Unlike classical scientific computational methods, such as finite element methods [14, 10, 17], in which solution accuracy and reliability are guaranteed under regularity conditions, standard DL methods are often far from providing reliable and accurate predictions for science and engineering applications. The reason is that though the approximation capability of deep learning, e.g. via Deep Neural Networks (DNN), is as good as classical methods in approximation theory [16, 23, 37, 26], DL accuracy is hardly attainable in general due to limitation in training. It has been shown that the training problem is highly nonlinear and non-convex, and that the gradient of loss functions can explode or vanish [22], thus possibly preventing any gradient-based optimization methods from reliably converging to a minimizer. Even when converged, the prediction of the (approximate) optimal deep learning model can be prone to over-fitting and can have poor generalization error.

Many data-driven inverse problems in science and engineering problems have limited experimental or observational data, e.g. due to the cost of placing sensors (e.g. digging an oil well can cost million of dollars) or the difficulties of placing sensors in certain regions (e.g. deep ocean bottoms). DL, by design, does not require physics, but data. This is the strength of DL. It is also the key limitation to science and engineering problems in which underlying physics needs to be respected and higher accuracy may be required. In this case, purely data-based DL approaches are prone to over-fitting and thus incapable of respecting the physics or providing the desired accuracy. Similar to least squares finite element methods [8], we can train a DNN solution constrained by the PDE residual as a regularization [54, 50, 52, 53, 66, 62, 35, 46]). Such an approach attempts to learn an approximate solution by making the L^2-norm of PDE residual small. While universal approximation results (see, e.g., [16, 23, 37, 26, 11]) could ensure any desired accuracy with a sufficiently large number of neurons, practical network architectures are moderate in both depth and width. Therefore, the accuracy of learning PDE solutions in function spaces can be limited.

We are interested in *parametrized PDEs*—that are pervasive in design, control, optimization, inference, and uncertainty quantification. Attempts using pure data-driven deep learning to learn the parameter to observable map have been explored (see, e.g., [29, 64, 47, 59, 48, 29, 57, 24, 56]). Approaches using autoencoder spirit that train a forward network first and then an inverse network in tandem [34, 39] or both of them simultaneously [20] have also been proposed. The work in [67] proposes to use a graph neural network to approximate forward solver and fully connected neural network to learn a regularization via the prior knowledge. Once trained, both networks are deployed in a Tikhonov-like regularization algorithm to obtain the

inverse solution. While successes are reported, generalization capability, and hence success, could be limited to regimes seen in the training data as the governing equations—containing most, if not all, information about the underlying physics—are not involved in the training.

In order to take into account the underlying problem, a natural direction is to deploy deep learning methods as surrogates for expensive or difficult components in traditional methods. Such a hybrid approach can enjoy the benefits of both sides. For example, learning regularizers to penalize certain undesirable features has been proposed for both inverse [32, 38] and imaging [2] problems. Once trained, these regularizers can be used in any traditional inverse or imaging methods. The main disadvantage of these approaches is that they may still experience the same computational expense as traditional methods when the forward map is the most computationally intensive part. Learning the forward map [33, 67, 4, 49] is thus desirable, though it may not be considered as a model-aware approach.

A logical alternative is thus to constrain the learning of inverse solution with the underlying governing equations and/or physics. The work in [1] proposes to partially learn the gradient of a Tikhonov functional and uses the learned gradient to perform a gradient-based optimization method for solving imaging problems. A natural extension of the physics-informed neural network framework [13, 51, 35, 36] is to train two networks, one for solutions and another for unknown parameters. In an attempt to mimic the traditional PDE-constrained approach, [18, 6] parametrize the unknown parameters using feed-forward neural networks whose weights/biases are then found by an optimization approach constrained by the Navier-Stokes equations and heat equations. These methods, however, may not be efficient as new observational data (corresponding to new unknown parameters) requires retraining. It is also not clear how to extend them to statistical inverse problems.

Learning inverse maps constrained by the underlying governing equations has also been investigated. The work in [45], similar to [18, 6], presents an autoencoder-like approach in which the encoder is the inverse map and the decoder is the numerical solutions of the underlying governing equations evaluated at observational points. The network weights/biases are found by minimizing the data misfit. Taking both the data misfit and the regularization into account as in the traditional Tikhonov inversion approach, [25] solves 1D seismic inversion methods with promising results. The beauty of this approach is that, once trained, the neural network can be deployed to approximately solve inverse problems in real-time.

The main contributions of this paper—a detailed extension of an approach set forth in [42]—is as follows. Unlike similar and independent work in [45, 18, 6, 25, 65], our model-constrained deep neural network approach (`mcDNN`) has a theoretical foundation, from which and numerical evidence, we infer that `mcDNN` may not be a good learning strategy for inverse problems as it could be biased by the training data, though it is interpretable compared to a purely data-driven counterpart. This motivates us to develop a new model-constrained deep learning approach, called `TNet`, designed to learn the Tikhonov inverse solution, and indeed it recovers Tikhonov regularized solutions for linear inverse problems and respects the governing equations exactly at the training points. Owing to the model-constrained design, `TNet` should generalize well for low data regimes and our numerical results verify this. We also propose to randomize the training data and rigorously justify randomization as an implicit regularization that could improve the generalization of the proposed deep-learning approaches. We provide comprehensive numerical results to support our developments for 1D deconvolution, inverse

heat conductivity, and inverse initial conditions for both time-dependent 2D Burgers' and 2D Navier-Stokes equations.

There are several limitations of our approaches. Firstly, the current formulation requires a differentiable solver, thereby incurring training and memory costs that are proportional to the computation cost of several forward solutions. This can be resolved by using a differential numerical library and we are working towards this direction. Another approach is to replace the differential solver with differential residual evaluation. We will report these approaches in future work. Secondly, although TNet is interpretable and has higher accuracy compared to purely-driven approaches, by design its accuracy can not exceed the traditional Tikhonov method. Therefore, it is not recommended for (sufficiently) large data regimes, and in that case alternative approaches such as [38, 2, 32] may be preferable as they could be more accurate than traditional Tikhonov approach. Lastly, while existing DNN approaches may not have a principled way to determine an appropriate regularization parameter, `TNet` regularization parameter can be obtained directly from Tikhonov regularization parameter. However, akin to the Tikhonov regularization framework (or any optimization/training approach with regularization), determining the optimal regularization parameter for `TNet` is a problem-dependent task and could be computationally intensive.

The paper is organized as follows. In section 2 we introduce nonlinear inverse problems, and a data-driven naive DNN (`nDNN`) approach. The goal of section 3 is to present a model-constrained DNN (`mcDNN`) approach designed to learn the inverse map while being constrained by the parameter-to-observable map of the underlying discretized PDE. Though `mcDNN` is interpretable, it could be biased toward training data. This leads us to develop `TNet`—a Tikhonov neural network—in section 4 that aims to learn the Tikhonov solver while removing unnecessary biases. We show that data randomization can make `TNet` not only more robust but also generalize better: thanks to the model-constrained training. In section 5 and the supplementary document, comprehensive numerical results supporting our developments are presented for 1D deconvolution, inverse heat conductivity, and inverse initial conditions for both time-dependent 2D Burgers' and 2D Navier-Stokes equations. We conclude the paper with future research directions in section 6. Practical implementation aspects of our proposed approaches and specifications of trainings are provided in the supplementary document.

2. Introduction to forward and inverse problems.

The following notations are used in the paper. Boldface lowercases are reserved for (column) vectors, and uppercase letters are for matrices. We denote by $\boldsymbol{u} \in \mathbb{R}^m$ the parameters sought in the inversion or the parameter of interest (PoI), by $\boldsymbol{w} \in \mathbb{R}^s$ the forward states, by $\mathcal{G} : \mathbb{R}^s \to \mathbb{R}^n$ the forward map (computing some observable quantity of interest), and by $\boldsymbol{y} \in \mathbb{R}^n$ the observations given by

$$\boldsymbol{y} := \mathcal{G}\left(\boldsymbol{w}\left(\boldsymbol{u}\right)\right) + \boldsymbol{\eta}, \tag{2.1}$$

where $\boldsymbol{\eta}$ is some additive observation noise. The parameter-to-observable (PtO) map is the composition of the forward map $\mathcal{G}$ and the states, i.e., $\mathcal{G} \circ \boldsymbol{w}$. However for simplicity of the exposition, we do not distinguish it from the forward map and thus we also write $\mathcal{G} : \mathbb{R}^m \ni \boldsymbol{u} \mapsto \mathcal{G}\left(\boldsymbol{u}\right) := \mathcal{G}\left(\boldsymbol{w}\left(\boldsymbol{u}\right)\right) \in \mathbb{R}^n$. The forward state is the solution of the forward equation

$$\mathcal{F}\left(\boldsymbol{u}, \boldsymbol{w}\right) = \boldsymbol{f}. \tag{2.2}$$

Assume that (2.2) is well-posed so that, for a given set of parameters $\boldsymbol{u}$, one can (numerically for example) solve for the corresponding forward states $\boldsymbol{w} = \boldsymbol{w}(\boldsymbol{u}) := \mathcal{F}^{-1}(\boldsymbol{f})$. In the forward problem, we compute observational data $\boldsymbol{y}$ via (2.1) given a set of parameter $\boldsymbol{u}$. In the inverse problem, we seek to determine the unknown parameter $\boldsymbol{u}$ given some observational data $\boldsymbol{y}$, that is, we wish to construct the inverse of $\mathcal{G}$. Since m is typically (much) larger than n for many practical problems, the parameter-to-observable map $\mathcal{G}$ is not invertible even when $\mathcal{G}$ is linear. The inverse task is thus ill-posed and notoriously challenging as a solution for $\boldsymbol{u}$ may not exist, even when it may, it is not unique nor stably depends on the data $\boldsymbol{y}$. An approximate solution is typically sought via (either deterministic or statistical) regularization.

Given the popularity of emerging machine learning, in particular deep neural networks (DNN), methods, we may attempt to apply a naive pure data-driven DNN (`nDNN`) to learn the (ill-posed) inverse of $\mathcal{G}$, e.g.,

$$\min_{\mathbf{b},W} \mathcal{L}_{\texttt{nDNN}} = \frac{1}{2}\left\|U - \Psi(Y, W, \mathbf{b})\right\|^2 + \frac{\alpha_1}{2}\|W\|^2 + \frac{\alpha_2}{2}\|\mathbf{b}\|^2, \tag{nDNN}$$

where Ψ is a DNN with weight matrix W and bias vector $\mathbf{b}$ and the last two terms are regularizations for weights and biases with nonnegative regularization parameters α_1 and α_2. Here, $Y \in \mathbb{R}^{n\times n_t}$ is the data matrix concatenating n_t observational data $\boldsymbol{y}^i$, $i = 1, \ldots, n_t$, and $U \in \mathbb{R}^{m\times n_t}$ is the parameter matrix concatenating the corresponding parameter vectors $\boldsymbol{u}^i$. This approach completely disregards the underlying mathematical model (2.1)-(2.2). Even for linear inverse problem—for example, $\mathcal{G}(\boldsymbol{u}) = G\boldsymbol{u}$ and there is no error in computing the data so that $Y = GU$—and linear DNN such as $\Psi = WY + B$, where $B := \mathbf{b}\mathbb{1}^T$, and thus the optimal weight W^0 and bias $\mathbf{b}^0$ for (`nDNN`) are given as

$$W^0 = U\left(I - \frac{1}{n_t+\alpha_2}\mathbb{1}\mathbb{1}^T\right)Y^T\left[Y\left(I - \frac{1}{n_t+\alpha_2}\mathbb{1}\mathbb{1}^T\right)Y^T + \alpha_1 I\right]^\dagger$$

$$\mathbf{b}^0 = \frac{1}{1+\alpha_2/n_t}\left(\overline{\boldsymbol{u}} - W^0\overline{\boldsymbol{y}}\right),$$

where $\overline{\boldsymbol{u}} := \frac{1}{n_t}U\mathbb{1}$ and $\dagger$ denotes the pseudo-inverse operation, it is not clear if the `nDNN` inverse solution

$$\boldsymbol{u}^{\texttt{nDNN}} = W^0\boldsymbol{y}^{obs} + \mathbf{b}^0.$$

provides an approximate solution to the original inverse problem

$$\min_{\boldsymbol{u}} \left\|\boldsymbol{y}^{obs} - G\boldsymbol{u}\right\|^2$$

in an interpretable sense. This is a disadvantage of pure data-driven approaches.

The data-driven nature of DNN could be claimed as an advantage. However, DNN can be considered as an "interpolation" method and thus can generalize well only for scenarios that have been seen in or are sufficiently close to the training data set $\{U, Y\}$. This implies a possible enormous amount of training data to learn the inverse of highly non-linear problems. In practical sciences and engineering problems, this extensive data regime is unfortunately

rarely the case due to the high cost of placing sensors or the difficulties in placing sensors in certain regions. *In order for a DNN to generalize well in insufficient/low data regimes, it should be equipped with information encoded in the forward model* (2.1)-(2.2) *that is not covered in the data set. Such a physics encoding also supplies meaningful interpretations to DNN inverse solutions as we shall show.* The question is how to inform DNN about the underlying models? In the following, we construct two *DNNs to learn the inverse of the PtO map* $\mathcal{G}$ *not only by information hidden in the training data but also by satisfying the forward equations exactly at the training points.*

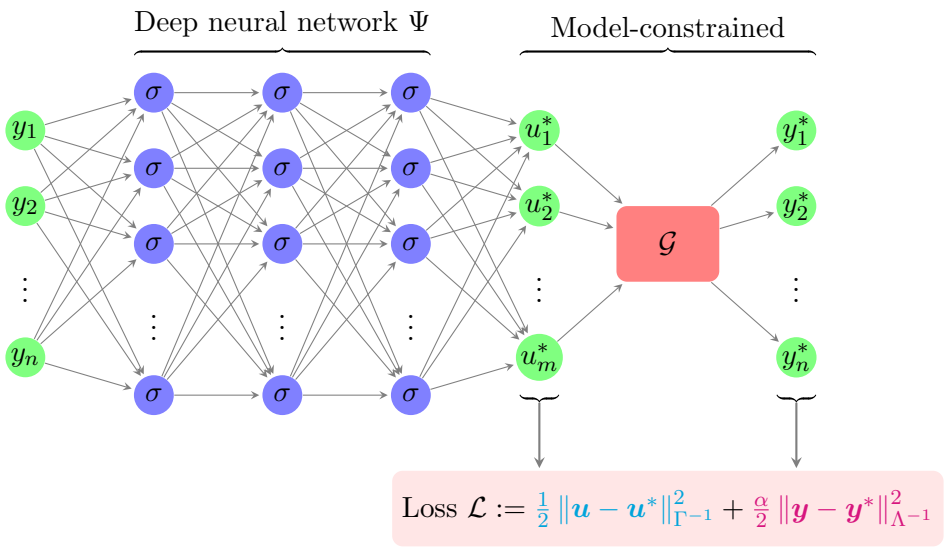

Figure 1: Model-constrained neural network architecture `mcDNN`. The observables $\boldsymbol{y}$ is fed into the neural network Ψ. The parameter $\boldsymbol{u}^*$ predicted by the network is pushed through the PtO map $\mathcal{G}$ to generate the corresponding predicted observations $\boldsymbol{y}^*$. Both predicted parameters and observations are compared with ground truth $\boldsymbol{u}$ and $\boldsymbol{y}$, respectively, to provide the mean-square error in the loss function $\mathcal{L}$.

3. Model-Constrained Deep Neural Network (`mcDNN`) for learning the inverse map. We propose to learn the inverse map via DNN constrained by the forward map as

$$\text{(mcDNN)} \qquad \min_{\mathbf{b},W} \mathcal{L}_{\texttt{mcDNN}} := \frac{1}{2}\left\|U - \Psi\left(Y, W, \mathbf{b}\right)\right\|^2_{\Gamma^{-1}} + \frac{\alpha}{2}\left\|Y - \mathcal{G}\left(\Psi\left(Y, W, \mathbf{b}\right)\right)\right\|^2_{\Lambda^{-1}},$$

where Ψ is a DNN learning the map from observable data $\boldsymbol{y}$ to parameter $\boldsymbol{u}$ with weight matrix W and bias vector $\mathbf{b}$. We have introduced Frobenius norm weighted by Γ^{-1} in the first term as

$$\left\|U - \Psi\left(Y, W, \mathbf{b}\right)\right\|^2_{\Gamma^{-1}} := \left\|\Gamma^{-\frac{1}{2}}\left(U - \Psi\left(Y, W, \mathbf{b}\right)\right)\right\|^2,$$

and similarly for the second term weighted by Λ^{-1}. Note that Λ is typically chosen as the covariance of the noise. The discussion of Γ is given in section 4. Unlike the naive

purely data-driven DNN approach (nDNN), the model-constrained (mcDNN) makes the DNN Ψ aware that the training data is generated by the forward map $\mathcal{G}$. This is done by requiring the output of the DNN—approximate unknown parameter $\boldsymbol{u}$ for a given data $\boldsymbol{y}$ as the input—when pushed through the forward model $\mathcal{G}$, reproduces the data $\boldsymbol{y}$. The model-aware term $\frac{\alpha}{2}\left\|Y-\mathcal{G}\left(\Psi\left(Y,W,\mathbf{b}\right)\right)\right\|_{\Lambda^{-1}}^2$ can be considered as a physics-aware regularization approach for mcDNN (compared to the non-physical regularizations in (nDNN)). The architecture of mcDNN is presented in Figure 1.

In order to shed light on our mcDNN approach let us choose a linear activation function such that the one-layer DNN model $\Psi\left(Y,W,\mathbf{b}\right)$ for leaning the inverse map can be written as $WY+B$, where $B:=\mathbf{b}\mathbb{1}^T$. We also assume that the forward map is linear. For linear inverse problem with linear DNN, the model-constrained training problem (mcDNN) becomes

$$\min_{\mathbf{b},W}\frac{1}{2}\left\|U-\left(WY+B\right)\right\|_{\Gamma^{-1}}^2+\frac{\alpha}{2}\left\|Y-G\left(WY+B\right)\right\|_{\Lambda^{-1}}^2. \tag{3.1}$$

Lemma 3.1. *The optimal solution W^I and $\mathbf{b}^I$ of the DNN training problem (3.1) satisfies*

$$\mathbf{b}^I=\left(\Gamma^{-1}+\alpha G^T\Lambda^{-1}G\right)^{-1}\left[\Gamma^{-1}\overline{\boldsymbol{u}}+\alpha G^T\Lambda^{-1}\overline{\boldsymbol{y}}-\left(\Gamma^{-1}\overline{U}\,\overline{Y}^\dagger+\alpha G^T\Lambda^{-1}\overline{Y}\,\overline{Y}^\dagger\right)\overline{\boldsymbol{y}}\right],$$

$$W^I=\left(\Gamma^{-1}+\alpha G^T\Lambda^{-1}G\right)^{-1}\left[\Gamma^{-1}\overline{U}\,\overline{Y}^\dagger+\alpha G^T\Lambda^{-1}\overline{Y}\,\overline{Y}^\dagger\right],$$

where $\overline{\boldsymbol{u}}:=\frac{1}{n_t}U\mathbb{1}$ and $\overline{\boldsymbol{y}}:=\frac{1}{n_t}Y\mathbb{1}$ are the column-average of the training parameters and data, $\overline{Y}:=Y-\overline{\boldsymbol{y}}\mathbb{1}^T$, and $\overline{U}:=U-\overline{\boldsymbol{u}}\mathbb{1}^T$.

Proof. Requiring the derivative of $\mathcal{L}_{\texttt{mcDNN}}$ in (3.1) with respect to $\mathbf{b}$ to vanish yields

$$\left(\Gamma^{-1}+\alpha G^T\Lambda^{-1}G\right)\mathbf{b}=\left[\Gamma^{-1}\left(U-WY\right)+\alpha G^T\Lambda^{-1}\left(Y-GWY\right)\right]\frac{\mathbb{1}}{n_t}. \tag{3.2}$$

Similarly, setting derivative of $\mathcal{L}_{\texttt{mcDNN}}$ with respect to W to zero gives

$$\Gamma^{-1}UY^T+\alpha G^T\Lambda^{-1}YY^T-\left(\Gamma^{-1}+\alpha G^T\Lambda^{-1}G\right)\left(\mathbf{b}\mathbb{1}^T\right)Y^T=\left(\Gamma^{-1}+\alpha G^T\Lambda^{-1}G\right)WYY^T. \tag{3.3}$$

Solving (3.2) to (3.3) for $\mathbf{b}$ and W we obtain

$$W=\left(\Gamma^{-1}+\alpha G^T\Lambda^{-1}G\right)^{-1}\left(\Gamma^{-1}\overline{UY}^\dagger+\alpha G^T\Lambda^{-1}\overline{YY}^\dagger\right), \tag{3.4}$$

and

$$\mathbf{b}=\left(\Gamma^{-1}+\alpha G^T\Lambda^{-1}G\right)^{-1}\left[\Gamma^{-1}\overline{\boldsymbol{u}}+\alpha G^T\Lambda^{-1}\overline{\boldsymbol{y}}-\left(\Gamma^{-1}\overline{U}\,\overline{Y}^\dagger+\alpha G^T\Lambda^{-1}\overline{Y}\,\overline{Y}^\dagger\right)\overline{\boldsymbol{y}}\right], \tag{3.5}$$

and this ends the proof. ∎

Corollary 3.2 (mcDNN is a Tikhonov solver). *For a given testing/observational data $\boldsymbol{y}^{obs}$, the mcDNN inverse solution $\boldsymbol{u}^{mcDNN}$ of (3.1) is given by*

$$\boldsymbol{u}^{mcDNN}=\left(\Gamma^{-1}+\alpha G^T\Lambda^{-1}G\right)^{-1}\left[\Gamma^{-1}\overline{\boldsymbol{u}}+\alpha G^T\Lambda^{-1}\overline{\boldsymbol{y}}+\left(\Gamma^{-1}\overline{U}\,\overline{Y}^\dagger+\alpha G^T\Lambda^{-1}\overline{Y}\,\overline{Y}^\dagger\right)\left(\boldsymbol{y}^{obs}-\overline{\boldsymbol{y}}\right)\right]$$

which is exactly the solution of the following Tikhonov regularized linear inverse problem

$$\min_{\boldsymbol{u}} \frac{1}{2}\left\|\boldsymbol{y}^{obs} - G\boldsymbol{u}\right\|^2_{\Lambda^{-1}} + \frac{1}{2\alpha}\left\|\boldsymbol{u} - \boldsymbol{u}_0^{mcDNN}\right\|^2_{\Gamma^{-1}},$$

where

$$\boldsymbol{u}_0^{mcDNN} = \overline{\boldsymbol{u}} + \overline{U}\,\overline{Y}^{\dagger}\left(\boldsymbol{y}^{obs} - \overline{\boldsymbol{y}}\right) - \alpha\Gamma G^T\Lambda^{-1}\left(I - \overline{Y}\,\overline{Y}^{\dagger}\right)\left(\boldsymbol{y}^{obs} - \overline{\boldsymbol{y}}\right). \tag{3.6}$$

The results of Corollary 3.2 shows that the `mcDNN` inverse solution $\boldsymbol{u}^{\texttt{mcDNN}}$ is equivalent to a Tikhonov-regularized inverse solution with a data-informed reference parameter $\boldsymbol{u}_0$ that depends on the training set $\{U, Y\}$ and the given observational data $\boldsymbol{y}^{obs}$. In other words, *the model-constrained deep learning `mcDNN` approach is interpretable in the sense that it provides data-informed Tikhonov-regularized inverse solutions.*

4. Tikhonov neural network (`TNet`) for learning the inverse map. We observe that the reference parameter $\boldsymbol{u}_0^{\texttt{mcDNN}}$ in Corollary 3.2 depends on the training data, and thus the model generalization depends on the amount of training data. In other words, `mcDNN` solution $\boldsymbol{u}^{\texttt{mcDNN}}$ could have a strong bias to the training data and may limit the generalization which is not desirable especially for scenarios that are not very close to the training ones. On the other hand, in the classical Tikhonov regularization framework, the reference parameter is fixed and independent of the observable data. From a statistical point of view, the reference parameter is typically the mean of the prior distribution of the parameter of interest (PoI) $\boldsymbol{u}$, which reflects the *a priori* belief on how the PoI should look like on average. Synergizing `mcDNN` and Tikhonov regularization ideas, we propose a Tikhonov neural network `TNet`—a semi-supervised model-constrained learning approach—where, unlike `mcDNN`, the unknown PoI predicted by the DNN Ψ are forced to be close a reference parameter $\boldsymbol{u}_0$ as

$$\min_{\mathbf{b},W} \mathcal{L}_{\texttt{TNet}} := \frac{1}{2}\left\|U_0 - \Psi\left(Y, W, \mathbf{b}\right)\right\|^2_{\Gamma^{-1}} + \frac{\alpha}{2}\left\|Y - \mathcal{G}\left(\Psi\left(Y, W, \mathbf{b}\right)\right)\right\|^2_{\Lambda^{-1}}, \tag{TNet}$$

where $U_0 = \boldsymbol{u}_0\mathbb{1}^T$ is the matrix whose each column is the reference parameter $\boldsymbol{u}_0$, and Γ is a chosen weight matrix appropriate for the problem under consideration.[1] Consequently, the architecture of `TNet` is the same as `mcDNN` in Figure 1 except with $\boldsymbol{u}$ replaced by $\boldsymbol{u}_0$. Applying Corollary 3.2 to `TNet` for linear inverse problem with linear DNN we have the following result.

Corollary 4.1 (`TNet` is a Tikhonov solver). *For a given testing/observational data $\boldsymbol{y}^{obs}$, the `TNet` inverse solution $\boldsymbol{u}^{TNet}$ is given by*

$$\boldsymbol{u}^{TNet} = \left(\Gamma^{-1} + \alpha G^T\Lambda^{-1}G\right)^{-1}\left[\Gamma^{-1}\boldsymbol{u}_0 + \alpha G^T\Lambda^{-1}\overline{\boldsymbol{y}} + \alpha G^T\Lambda^{-1}\overline{Y}\,\overline{Y}^{\dagger}\left(\boldsymbol{y}^{obs} - \overline{\boldsymbol{y}}\right)\right] \tag{4.1}$$

which is exactly the solution to the following Tikhonov regularized linear inverse problem

$$\min_{\boldsymbol{u}} \frac{1}{2}\left\|\boldsymbol{y}^{obs} - G\boldsymbol{u}\right\|^2_{\Lambda^{-1}} + \frac{1}{2\alpha}\left\|\boldsymbol{u} - \boldsymbol{u}_0^{TNet}\right\|^2_{\Gamma^{-1}},$$

where

$$\boldsymbol{u}_0^{TNet} = \boldsymbol{u}_0 - \alpha\Gamma G^T\Lambda^{-1}\left(I - \overline{Y}\,\overline{Y}^{\dagger}\right)\left(\boldsymbol{y}^{obs} - \overline{\boldsymbol{y}}\right).$$

[1] In the Bayesian setting, $\boldsymbol{u}_0$ and Γ are corresponding to the mean and covariance of a Gaussian prior.

Two observations are in order. First, Corollary 4.1 shows that the TNet inverse solution $\boldsymbol{u}^{\texttt{TNet}}$ is exactly the Tikhonov-regularized inverse solution with the true prior mean $\boldsymbol{u}_0$ as the reference parameter provided that the observation data Y is full row rank. This holds, for example, when the number of independent data is at least the same as the number of observations. Even when this happens, mcDNN solution in Corollary 3.2 does not coincide with the Tikhonov solution as the first two terms on the right-hand side of (3.6) only reduce to $\boldsymbol{u}_0$ in the limit of infinite training data (via the law of large numbers). Second, training data for PoI $\boldsymbol{u}$ is not needed (thanks to the semi-supervised learning nature of TNet). This is particularly useful when we like to use actual observational data in training.

The next result is a highlight of our method in that our model-constrained approaches satisfy the governing equation exactly at the training points. The aspects of practical implementation are provided in section SM1.

Remark 4.2 (Exactly satisfying the governing equations at training points). Note that both mcDNN and TNet inverse solutions satisfy the governing equations exactly at all training points. Since mcDNN and TNet share the same model-constrained term, we only need to provide the proof for TNet. Due to the assumption that (2.2) is well-posed, we can write the TNet training problem (TNet) equivalently as

$$\min_{\mathbf{b},W} \mathcal{L}_{\texttt{TNet}} := \frac{1}{2}\left\|U_0 - \Psi\left(Y, W, \mathbf{b}\right)\right\|^2_{\Gamma^{-1}} + \frac{\alpha}{2}\sum_{i=1}^{n_{\mathrm{t}}}\left\|\boldsymbol{y}^i - \mathcal{G}\left(\boldsymbol{w}^i\right)\right\|^2_{\Lambda^{-1}}, \tag{4.2}$$

subject to

$$\mathcal{F}\left(\Psi\left(\boldsymbol{y}^i, W, \mathbf{b}\right), \boldsymbol{w}^i\right) = \boldsymbol{f}, \quad i = 1, \ldots, n_{\mathrm{t}},$$

which clearly shows that in fact TNet formulation (TNet) is a hard-constrained optimization problem that ensures the forward equation (2.2) to be satisfied exactly at all the training points during the training.

We now show that data randomization enhances not only the generalization of TNet solution but also its robustness to observational noise. To begin, we randomize a generic data vector $\boldsymbol{y}$, e.g. one column of Y, as follows

$$\tilde{\boldsymbol{y}} = \boldsymbol{y} + \boldsymbol{\varepsilon}, \tag{4.3}$$

where a Gaussian noise vector $\boldsymbol{\varepsilon} \sim \mathcal{N}\left(0, \lambda^2 I\right)$ with variances λ^2 is added to the data. We emphasize that the following arguments also hold for any random noise vector with independent components, each of which is a random variable with zero mean and variances λ^2. Let $\mathbb{E}\left[\cdot\right]$ denote the expectation with respect to $\boldsymbol{\varepsilon}$. Following [5], for a generic loss function $\mathcal{L}\left(\tilde{\boldsymbol{y}}\right)$, we perform the Taylor expansion around $\boldsymbol{y}$ up to second order to obtain

$$\mathbb{E}\left[\mathcal{L}\left(\tilde{\boldsymbol{y}}\right)\right] = \mathcal{L}\left(\boldsymbol{y}\right) + \mathbb{E}\left[\left.\frac{\partial \mathcal{L}}{\partial \boldsymbol{y}}\right|_{\boldsymbol{y}} \boldsymbol{\varepsilon}\right] + \frac{1}{2}\mathbb{E}\left[\boldsymbol{\varepsilon}^T \left.\frac{\partial^2 \mathcal{L}}{\partial \boldsymbol{y}^2}\right|_{\boldsymbol{y}} \boldsymbol{\varepsilon}\right] + \mathbb{E}\left[o\left(\left\|\boldsymbol{\varepsilon}\right\|^2\right)\right] \tag{4.4}$$

where we have used sufficient small noise variance λ^2 so that the high-order term $o\left(\left\|\boldsymbol{\varepsilon}\right\|^2\right)$, using the standard "small o" notation, is negligible.

For training data set with n_t samples, we randomize each sample as $\tilde{\boldsymbol{y}}^i = \boldsymbol{y}^i + \boldsymbol{\varepsilon}^i, i = 1, \ldots, n_t$, where $\boldsymbol{\varepsilon}_i \sim \mathcal{N}\left(0, \lambda_i^2 I\right)$. Note that we can use different noise levels for data randomization. In that case, the (TNet) loss becomes

$$\mathcal{L}_{\texttt{TNet}}^{\text{rand}} = \sum_{i=1}^{n_t} \underbrace{\frac{1}{2}\left\|\boldsymbol{u}_0 - \Psi(\tilde{\boldsymbol{y}}^i)\right\|_{\Gamma^{-1}}^2 + \frac{\alpha}{2}\left\|\tilde{\boldsymbol{y}}^i - \mathcal{G}\left(\Psi\left(\tilde{\boldsymbol{y}}^i\right)\right)\right\|_{\Lambda^{-1}}^2}_{=:\mathcal{J}(\tilde{\boldsymbol{y}}^i)}. \tag{4.5}$$

Replacing $\mathcal{L}$ with $\mathcal{J}(\tilde{\boldsymbol{y}}_i)$ in (4.4) yields

$$\mathbb{E}\left[\mathcal{J}\left(\tilde{\boldsymbol{y}}^i\right)\right] \approx \mathcal{J}\left(\boldsymbol{y}^i\right) + \lambda_i^2\left(\mathcal{P}_1^i + \mathcal{P}_2^i + \mathcal{P}_3^i + \mathcal{P}_4^i\right), \tag{4.6}$$

where

$$\mathcal{J}\left(\boldsymbol{y}^i\right) = \frac{1}{2}\left\|\boldsymbol{u}_0 - \Psi(\boldsymbol{y}^i)\right\|_{\Gamma^{-1}}^2 + \frac{\alpha}{2}\left\|\boldsymbol{y}^i - \mathcal{G}\left(\Psi\left(\boldsymbol{y}^i\right)\right)\right\|_{\Lambda^{-1}}^2, \tag{4.7}$$

and the *induced penalty* terms are given by

$$\begin{aligned}
\mathcal{P}_1^i &= \frac{1}{2}\mathbf{Tr}\left[\left(\nabla_{\boldsymbol{y}}\Psi\left(\boldsymbol{y}^i\right)\right)^T \Gamma^{-1}\left(\nabla_{\boldsymbol{y}}\Psi\left(\boldsymbol{y}^i\right)\right)\right], \\
\mathcal{P}_2^i &= \frac{\alpha}{2}\mathbf{Tr}\left[\left(\nabla_{\boldsymbol{y}}\left[\boldsymbol{y}^i - \mathcal{G}\circ\Psi\left(\boldsymbol{y}^i\right)\right]\right)^T \Lambda^{-1}\left(\nabla_{\boldsymbol{y}}\left[\boldsymbol{y}^i - \mathcal{G}\circ\Psi\left(\boldsymbol{y}^i\right)\right]\right)\right], \\
\mathcal{P}_3^i &= \frac{1}{2}\mathbf{Tr}\left[\nabla_{\boldsymbol{y}}^2\Psi\left(\boldsymbol{y}^i\right)\odot\Gamma^{-1}\left(\Psi\left(\boldsymbol{y}^i\right) - \boldsymbol{u}_0\right)\right], \\
\mathcal{P}_4^i &= \frac{\alpha}{2}\mathbf{Tr}\left[\nabla_{\boldsymbol{y}}^2\left[\boldsymbol{y}^i - \mathcal{G}\circ\Psi\left(\boldsymbol{y}^i\right)\right]\odot\Lambda^{-1}\left(\boldsymbol{y}^i - \mathcal{G}\circ\Psi\left(\boldsymbol{y}^i\right)\right)\right],
\end{aligned}$$

in which $\mathbf{Tr}\left(\cdot\right)$ is the trace operator, and $\odot$ denotes the dot product of a third-order tensor and a vector. It can be observed that the training loss with randomized data is the sum of the original loss plus four induced regularization terms. $\mathcal{P}_1^i$ is non-negative and promotes the smoothness of the neural network. The second term of (4.7) ensures that Ψ is close to the right inverse of $\mathcal{G}$, and $\mathcal{P}_2^i$ strengthen this closeness by forcing the derivative of the $\mathcal{G}\circ\Psi$ to the identity. These two effects together behave like a Hermite interpolation in which not only the function values but also the derivatives are required to be matched closely at the training points. The two terms in (4.7) make $\Psi\left(\boldsymbol{y}^i\right) - \boldsymbol{u}_0$ and $\boldsymbol{y}^i - \mathcal{G}\circ\Psi\left(\boldsymbol{y}^i\right)$ necessary small, and as a result, $\mathcal{P}_3^i$ and $\mathcal{P}_4^i$ can be dominated by $\mathcal{P}_1^i$ and $\mathcal{P}_2^i$, respectively. It is interesting to see that $\mathcal{P}_3^i$ and $\mathcal{P}_4^i$ can encourage the second derivatives (and hence extra smoothness) of the neural network Ψ and $I - \mathcal{G}\circ\Psi$ to be small. In other words, the beauty of data randomization is that it can promote a $\mathcal{H}^2$-Sobolev-like Tikhonov regularization for the neural network Ψ via (4.7), $\mathcal{P}_1^i$, and $\mathcal{P}_3^i$. Moreover, it can further enforce Ψ to be the same as the right inverse of the PtO map $\mathcal{G}$ up to second derivatives via (4.7), $\mathcal{P}_2^i$, and $\mathcal{P}_4^i$.

Accounting for the data randomization for all training data we can—after taking the expectation with the random noise $\boldsymbol{\varepsilon}^i$, $i = 1, \ldots, n_t$—write (4.5) as

$$\mathbb{E}\left[\mathcal{L}_{\texttt{TNet}}^{\text{rand}}\right] \approx \mathcal{L}_{\texttt{TNet}} + \frac{1}{2}\sum_{i=1}^{n_t}\lambda_i^2\left(\mathcal{P}_1^i + \mathcal{P}_2^i + \mathcal{P}_3^i + \mathcal{P}_4^i\right). \tag{4.8}$$

Thus, on average, the TNet loss $\mathcal{L}_{\texttt{TNet}}^{\text{rand}}$ with randomized data is approximately the sum of the original TNet loss (without randomization) plus four regularization terms for each training data point. These induced regularization terms play a vital role in stimulating the robustness and accuracy of the neural network. Indeed, without data randomization, the TNet loss (TNet) simply requires the neural network outputs to be close to the parameter data via the data misfit (the first) term, and the neural network, when pushed through the PtO map, resembles the observational data via the model-constrained (the second) term. Whereas, randomizing the data enforces not only the smoothness of the neural network Ψ up to second order derivative (through the first term) but also the agreement of the neural network and the right inverse of the PtO map $\mathcal{G}$ up to second order derivatives (through the model-constrained term). Let us summarize the above result in the following theorem.

Theorem 4.3. *Let* $\tilde{\boldsymbol{y}}^i = \boldsymbol{y}^i + \boldsymbol{\varepsilon}^i, i = 1, \ldots, n_t$, *where* $\boldsymbol{\varepsilon}_i \sim \mathcal{N}\left(0, \lambda_i^2 I\right)$. *Then*

$$\mathbb{E}\left[\mathcal{L}_{TNet}^{rand}\right] = \mathcal{L}_{TNet} + \frac{1}{2}\sum_{i=1}^{n_t} \lambda_i^2 \left(\mathcal{P}_1^i + \mathcal{P}_2^i + \mathcal{P}_3^i + \mathcal{P}_4^i\right) + \sum_{i=1}^{n_t} \mathbb{E}\left[o\left(\left\|\boldsymbol{\varepsilon}^i\right\|^2\right)\right]. \tag{4.9}$$

Remark 4.4. Note that $\boldsymbol{y}^i$ are not necessarily different from each other. However, the Hermite interpolation analogy tells us that we should have as many distinct baseline training points as possible for good generalization. It turns out that we just need a small number of distinct training points to have accurate results, as numerically shown in section 5. The above randomization approach also holds for nDNN and mcDNN approaches. Indeed, in the the final expression (4.8) we simply replace $\mathcal{L}_{\texttt{TNet}}$ by $\mathcal{L}_{\texttt{mcDNN}}$ (see (mcDNN)) and $\boldsymbol{u}_0$ by $\boldsymbol{u}^i$ in $\mathcal{P}_3^i$ for mcDNN. Similarly for nDNN, we replace $\mathcal{L}_{\texttt{TNet}}$ by $\mathcal{L}_{\texttt{nDNN}}$ (see (nDNN)) and remove $\mathcal{P}_2^i$ and $\mathcal{P}_4^i$.

5. Numerical results.

Noise realization. For all numerical results, we choose $\lambda = \delta \max(\boldsymbol{y})$ for all λ_i in (4.9), where δ denotes the relative noise level.

Data generation and training. For non-linear problems in subsection 5.1, subsection SM2.2 and subsection 5.2, we use a shallow neural network having one hidden layer with 5000 ReLU neurons. We verified that a dense feed-forward neural network architecture with multiple layers could provide comparable results but with large training data sets. In small data regimes, i.e. 100 samples, deep networks perform poorly due to the vanishing gradient problem and/or the bias-variance trade-off problem. Moreover, training a deep learning network faces further challenges [19, 58] that are beyond the scope of this paper. We thus focus on neural networks with a single hidden layer and this is sufficient to demonstrate the proposed TNet framework. Regarding optimization algorithm, the default ADAM [28] optimizer in JAX [9] is used. In all numerical results, weights and biases of the neural network are initialized by standard Gaussian distribution and a zero vector, respectively, using the same random seed. Therefore, we begin the training process with the same network for all cases.

In order to be fair, within any comparison we use the same random seeds for noise. To ensure that more training data can offer more information, the training data set is generated in a nested manner, e.g., $n_t = 50 \subset n_t = 100 \subset n_t = 200 \subset \ldots$, and so on, where n_t denotes the number of training samples. For any testing, except for the linear deconvolution problem in which we use 200 testing samples, a test data set of 500 samples is used to compare

approaches. The Tikhonov inverse solutions are obtained by the default BFGS algorithm [43] in `Jax` [9]. A summary of training parameters is presented in Table 1.

Accuracy metric. To estimate the accuracy of each approach, we compute average relative errors from $M = \{200, 500\}$ unseen random samples: the first based on pointwise values and the second on Euclidean norm of the physical parameter vector (which is a function of inverse parameter $\boldsymbol{u}$: see (5.3) (5.5) (SM2.1)) as follows

$$\text{Err}_j = \frac{1}{M}\sum_{i=1}^{M} \frac{\left(\omega_j^{i,\text{pred}} - \omega_j^{\text{true}}\right)^2}{\|\omega^{\text{true}}\|^2/m} \times 100 \quad (\%), \tag{5.1}$$

and

$$\text{Err} = \frac{1}{M}\sum_{i=1}^{M} \frac{\left\|\omega^{i,\text{pred}} - \omega^{\text{true}}\right\|^2}{\|\omega^{\text{true}}\|^2} = \frac{1}{m}\sum_{j=1}^{m} \text{Err}_j \quad (\%), \tag{5.2}$$

where superscript i denotes the ith sample, subscript j denotes the jth component of the vector under consideration, and m is the number of spatial grid points. Here, "pred" stands for the solution predicted by the neural network, and "true" for the synthetic ground truth parameters.

Training parameters. Table 1 summarizes the specifications for neural network architectures, training settings, testing data sets, etc.

Table 1: Summary of training parameters for `nDNN`, `mcDNN` and `TNet` for nonlinear inverse problems in subsection 5.1, subsection SM2.2 and subsection 5.2.

Network	Architecture	1 layer with 5000 neurons
	Activation function	ReLU
	Weight initializer	$\mathcal{N}(0, 0.02)$
	Bias initializer	$\mathbf{0}$
	Random seed	100
Training	Optimizer	`ADAM`
	Learning rate	10^{-3}
	Batch size	$= \begin{cases} n_t & \text{if } n_t \le 500 \\ 500 & \text{otherwise} \end{cases}$
Data	Train data	$n_t = p \subset n_t = q, \quad \text{if } p < q \text{ and } p, q \in \mathbb{N}$
	Test data	500 samples (drawn independently)
	Train random seed	18
	Test random seed	28
Precision		Double precision

5.1. 2D heat conductivity inverse problem. The heat equation we consider is the following

$$\begin{aligned} -\nabla \cdot (e^{\omega} \nabla y) &= 20 \quad \text{in } \Omega = (0,1)^2 \\ y &= 0 \quad \text{on } \Gamma^{\text{ext}} \\ \mathbf{n} \cdot (e^{\omega} \nabla y) &= 0 \quad \text{on } \Gamma^{\text{root}}, \end{aligned}$$

where ω is the the conductivity field, y is the temperature field, and $\mathbf{n}$ is the unit outward normal vector on Neumann boundary part Γ^{root} . Figure 2 shows the domain (left subfigure) and a 16×16 mesh (right subfigure) together with the locations of 10 observational points of the state y. In this problem, we are interested in reconstructing the conductivity field given a set of 10 pointwise observations.

Generating train and test data sets. We start with drawing the parameter conductivity samples via a truncated Karhunen-Loéve expansion

$$\omega(x) = \sum_{i=1}^{n} \sqrt{\lambda_i} \phi_i(x) u_i, \quad x \in [0,1]^2, \tag{5.3}$$

where (λ_i, ϕ_i) are the eigenpairs of the following two-point correlation function [15]:

$$\mathcal{C}(x_1, x_2) = \exp\left(-\frac{\|x_1 - x_2\|_1}{\beta} \right) \tag{5.4}$$

where $\|\cdot\|_1$ is the 1-norm on $\mathcal{R}^2$, $\beta = 0.02$ is the correlation length. Here, $\mathbf{u} = (u_i)_{i=1}^n \sim \mathcal{N}(0, I)$ is a standard Gaussian random vector. It should be noted that, rather than directly inverting for the physical parameter ω, we reconstruct the coefficient vector $\boldsymbol{u}$. Specifically, we select $n = 15$ eigenvectors corresponding to the first 15 largest eigenvalues. For each sample, we discretize ω and we solve the heat equation for the temperature $\boldsymbol{y}$ by finite element method. Observations are obtained by extracting values of the temperature field at 10 observational points, which are then corrupted with additive Gaussian noise with a noise level of $\delta = 0.5\%$. Figure 3 displays five different pairs of samples of the log conductivity field ω and the corresponding temperature field in the data set $d5$. Note that we generate test pairs $(\omega, \boldsymbol{y})$ using the same process.

Next, we consider two cases of train data for learning the inverse map from observations to conductivity. Case I: Full base, i.e., n_t distinct training samples are used; and Case II: we first pick a number of distinct baseline samples n_b smaller than n_t, and then replicate and randomize them to obtain n_t samples for the train data set. For each case, the average relative error in (5.2) is computed with 500 true test samples for `nDNN`, `mcDNN` and `TNet`, and is compared to the relative error of the Tikhonov regularization approach.

Case I: Training with full data sets $n_b = n_t \in \{50, 100, 200\}$. We train `nDNN`, `mcDNN` and `TNet` networks using three different full training data bases, $n_t = 50 \subset n_t = 100 \subset n_t = 200$ and present the smallest errors in Table 2. As can be seen, larger data sets provide more accurate inverse maps. In particular, the average smallest relative errors for `nDNN` for these training sets are 60.41%, 50.69% and 49.07% which are higher than 57.27%, 50.39% and 48.40%, respectively, for `mcDNN`. With the smallest errors of 45.98%, 45.35% and, 44.98%,

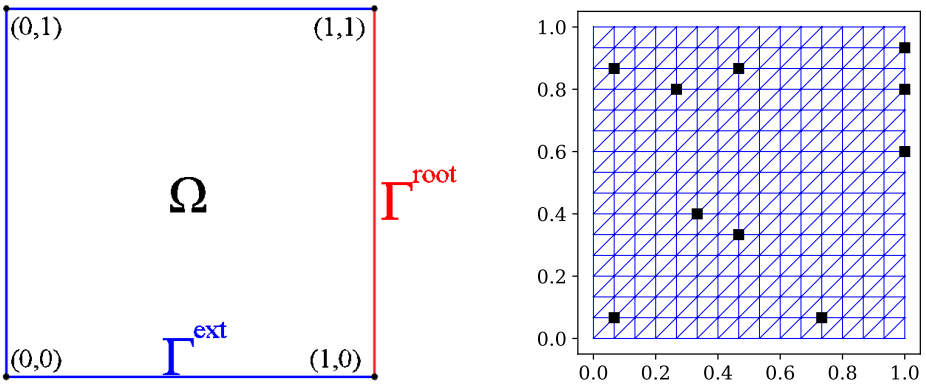

Figure 2: **2D heat conductivity inverse problem.** *Left figure* the domain and the boundaries; *Right figure* A 16×16 finite element mesh and 10 observational locations.

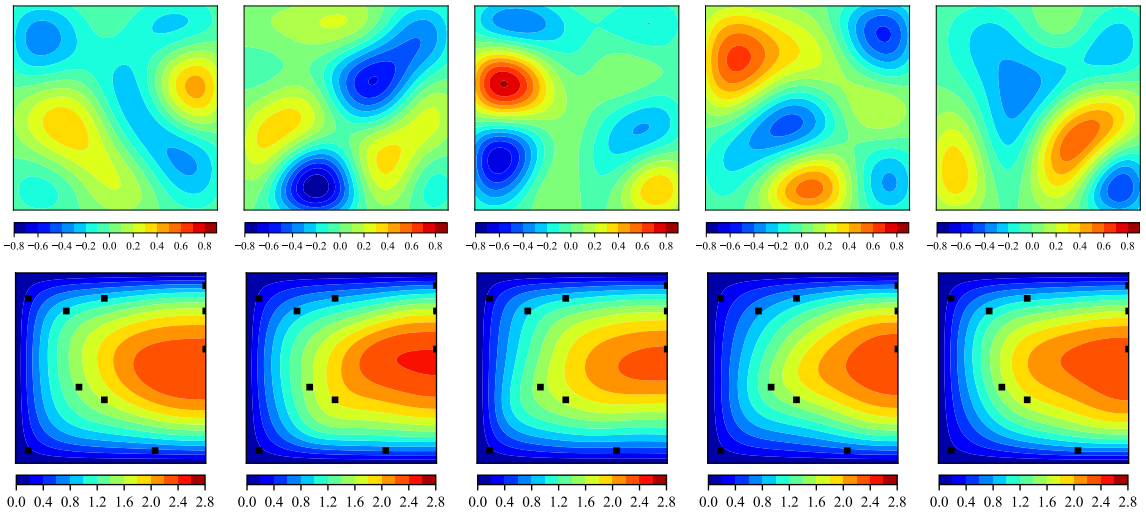

Figure 3: **2D heat conductivity inverse problem.** 5 distinct training pair samples in data set $d5$. *Top row:* the log conductivity field. *Bottom row:* The corresponding temperature field and 10 observations points

correspondingly, `TNet` outperforms `nDNN` and `mcDNN` by a significant margin, and is similar to Tikhonov (TIK) approach. It is not surprising since `TNet` approach is designed to learn Tikhonov method, as discussed in section 4. This is further confirmed by the fact that while regularization parameters for `mcDNN`, `TNet`, and Tikhonov approaches are the same, namely $\alpha = 8000$, only `TNet` and Tikhonov solutions agree well with each other for a wide range of regularization parameters, as shown in Figure 4. On the contrary, a data-driven approach such as `nDNN` requires sufficient training data (more than 100 for this case as Figure 4 indicates) to

provide a reasonable solution. We note that mcDNN is not much more accurate than nDNN for this example, perhaps due to strong bias from the data as suggested by Corollary 3.2.

The preceding discussion also alludes to an important point. In particular, identifying a good approximation of the optimal regularization parameter plays a vital part in TNet performance. This can be accomplished by finding a good regularization parameter for the Tikhonov approach and using it for TNet. The subject of determining a suitable regularization parameter has been studied extensively in the literature using various approaches including the Morozov discrepancy principle, L-curve, and cross-validation [41, 61, 63]. The numerical results in Figure 4 show that TNet and mcDNN results are robust in accuracy for a sufficiently large neighborhood around the optimal Tikhonov regularization parameter, and thus a reasonable regularization parameter is sufficient for TNet and mcDNN methods. Another important point that we show in the deconvolution subsection SM2.1 is that the optimal regularization parameter for TNet and mcDNN are numerically independent of training data sets, while it varies drastically for nDNN method. This implies TNet and mcDNN are more robust and reliable than nDNN.

Table 2: **2D heat conductivity inverse problem, Case I.** The average relative error (5.2) over 500 test samples obtained by nDNN (optimal α varies depending on the data set), mcDNN ($\alpha = 8000$), TNet ($\alpha = 8000$) with nested data sets $n_t = 50 \subset n_t = 100 \subset n_t = 200$, and Tikhonov (TIK) with $\alpha = 8000$.

	nDNN	mcDNN	TNet	TIK
$n_t = 50$	60.41	57.27	45.98	44.99
$n_t = 100$	50.69	50.39	45.35	
$n_t = 200$	49.07	48.40	44.98	

Case II: Training with $n_b = 20 < n_t \in \{60, 100, 200, 1000, 2000, 5000\}$. We now investigate how the data augmentation via randomization performs with nDNN, mcDNN and TNet. In particular, 20 noise-free baseline data pairs are replicated to create N samples of training data sets ranging from $n_t = 20$ to $n_t = 5000$, which are then randomized with 2% additive white noise. Table 3 shows the average relative error (5.2) of the test data set obtained by nDNN, mcDNN and TNet. In the first row are the results for the baseline case with $n_t = n_b = 20$ and this is used as the reference for the other rows. It can be seen that data randomization and augmentation, though regularizes the smoothness of the network, negligibly improves the accuracy of nDNN. Clearly, nDNN is not equipped with the forward map and completely depends on the limited information given in the baseline data. On the contrary, the accuracy for mcDNN is improved by about 10% for $n_t \geq 1000$. This is expected as Theorem 4.3 shows that randomization, via the model-constrained term, promotes the network solution to be the right inverse of the forward map up to second order. However, mcDNN's accuracy level saturates with $n_t = 1000$ and is still significantly higher than the Tikhonov approach (the last column). This is again due to data-dependent regularization nature (see Corollary 3.2), and hence biasing to the training data, of the mcDNN approach despite of the effectiveness of model-constrained term. Unlike nDNN and mcDNN approaches, TNet results are much more

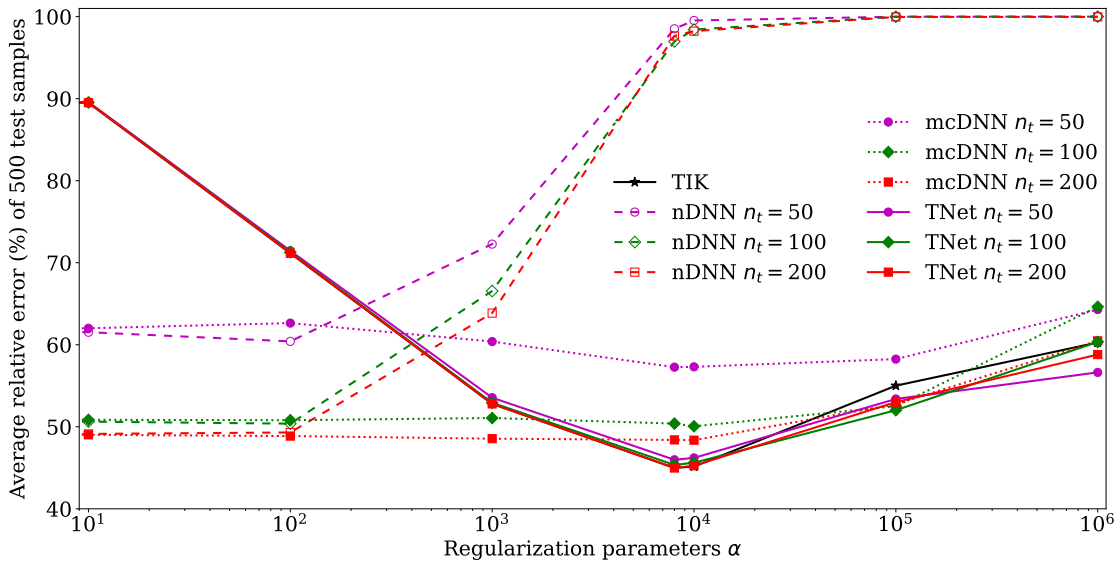

Figure 4: **2D heat conductivity inverse problem, Case I.** The average relative error (5.2) over 500 test samples with nested data sets $n_t = 50 \subset n_t = 100 \subset n_t = 200$. The comparisons are done for `nDNN` (dashed curves), `mcDNN` (dotted curves), `TNet` (colored solids curves), and Tikhonov (TIK: black curve) over a wide range of regularization parameter values.

accurate regardless of any considered value of n_t. Furthermore, they seem to approach the Tikhonov accuracy as n_t increases from 20 to 5000. In particular, `TNet` needs only about 100 samples replicated and randomized from $n_b = 20$ distinct baseline samples to learn an inverse map as nearly accurate as the Tikhonov solution.

Figure 5 shows the pointwise average error over 500 test samples (see (5.1)) for `nDNN`, `mcDNN`, `TNet`, and Tikhonov (TIK) approaches for $n_b = 20$ and $n_t = 200$. While `nDNN` and `mcDNN` have a high level of error, `TNet` has a similar error as the TIK solver in both values and patterns. For all these methods, we show in Figure 6 the reconstructed conductivities from a new unseen noisy data for $n_b = 20$ and $n_t = 200$. The synthetic ground truth conductivity and the corresponding temperature distribution are also presented for reference in the middle column. Again, the `TNet` inverse solution is in good agreement with the Tikhonov one, and thus with the ground truth, while `nDNN` and `mcDNN` yield quite inaccurate reconstructions.

How many baseline pairs are sufficient for `TNet`? For this problem we numerically study how many distinct baseline pairs are needed to achieve a reasonably accurate inverse solution from the `TNet` approach. Figure 7 shows that $n_b = 5$ baseline pairs are sufficient when $n_t \geq 1000$. For example, with $n_b = 5$ and $n_t = 1000$, `TNet` achieves a relative error of approximately 46.7% compared to 44.99 % of the Tikhonov solution. We also observe that, given an inadequate number of distinct baseline pairs, i.e., one or two, it is challenging to learn a highly accurate inverse operator even with data augmentation, randomization, and large n_t due to lacking information. This can be seen through the second term in (4.8). Indeed, in

Table 3: **2D heat conductivity inverse problem, Case II.** The average relative error (5.2) for `nDNN` (optimal α varies depending on the data set), `mcDNN` ($\alpha = 8000$), `TNet`($\alpha = 8000$), and Tikhonov (TIK) ($\alpha = 8000$) over 500-sample test data set obtained by training with $n_b = 20$ baseline data pairs.

	`nDNN`	`mcDNN`	`TNet`	TIK
$n_t = 20$	89.66	77.61	55.56	
$n_t = 60$	86.87	77.02	47.35	
$n_t = 100$	87.19	72.58	46.27	
$n_t = 200$	89.59	71.81	46.01	44.99
$n_t = 1000$	88.67	69.68	45.16	
$n_t = 2000$	88.31	69.72	45.23	
$n_t = 5000$	88.55	69.71	45.11	

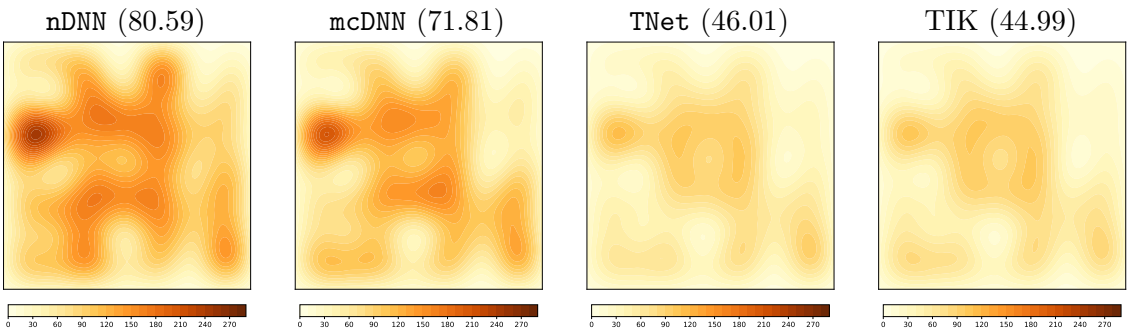

Figure 5: **2D heat conductivity inverse problem, Case II.** The distribution of average relative pointwise error (5.1) for `nDNN`, `mcDNN`, `TNet`, and Tikhonov (TIK) over 500 test samples obtained with $n_b = 20$ and $n_t = 200$. The numbers in the parentheses are the average error (5.2) incurred by these methods.

this case, we have

$$\frac{1}{2}\sum_{i=1}^{n_t}\lambda_i^2\left(\mathcal{P}_1^i+\mathcal{P}_2^i+\mathcal{P}_3^i+\mathcal{P}_4^i\right)=\frac{n_t}{2n_b}\sum_{i=1}^{n_b}\lambda_i^2\left(\mathcal{P}_1^i+\mathcal{P}_2^i+\mathcal{P}_3^i+\mathcal{P}_4^i\right),$$

and thus the induced regularizations are active only at the distinct baseline samples. For small baseline samples, there is simply not enough information for `TNet` to perform well. This again agrees with the Hermite interpolation analogy discussed in section 4.

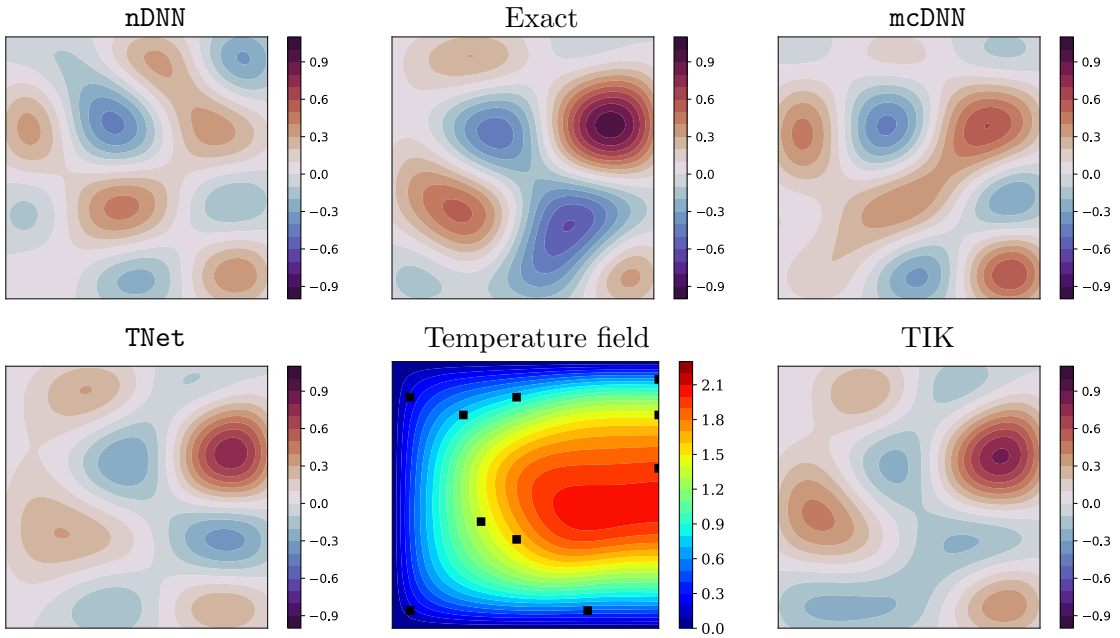

Figure 6: **2D heat conductivity inverse problem, Case II.** Predicted (reconstructed) heat conductivities for an unseen noisy test sample obtained by `nDNN`, `mcDNN`, and `TNet` neural networks along with the Tikhonov reconstruction for $n_b = 20$ and $n_t = 200$. Shown in the middle column are the synthetic ground truth (Exact) conductivity and the corresponding temperature field for reference.

5.2. 2D Navier-Stokes equation. The vorticity form of 2D Navier-Stokes equation for viscous and incompressible fluid [33] is written as

$$\begin{aligned} \partial_t \omega(x,t) + v(x,t)\cdot\nabla\omega(x,t) &= \nu\Delta\omega(x,t) + f(x), && x \in (0,1)^2, t \in (0,T] \\ \nabla\cdot v(x,t) &= 0, && x \in (0,1)^2, t \in (0,T] \\ \omega(x,0) &= \omega_0(x), && x \in (0,1)^2 \end{aligned}$$

where $v \in (0,1)^2 \times (0,T]$ is the velocity field, $\omega = \nabla \times v$ is the vorticity, ω_0 is the initial vorticity, $f(x) = 0.1\left(\sin\left(2\pi\left(x_1+x_2\right)\right) + \cos\left(2\pi\left(x_1+x_2\right)\right)\right)$ is the forcing function, and $\nu = 10^{-3}$ is the viscosity coefficient. The spatial domain is discretized with 32×32 uniform mesh, while the time horizon $t \in (0,10)$ is subdivided into 1000 time steps with $\Delta t = 10^{-2}$. We target to reconstruct the initial vorticity ω_0 from the measurements of vorticity at 20 observed points at the final time $T = 10$.

Generating train and test data sets. To generate data pairs of $(\omega, \boldsymbol{y})$, we draw samples of $\omega(x,0)$ using the truncated Karhunen-Loève expansion

$$\omega(x,0) = \sum_{i=1}^{24} \sqrt{\lambda_i}\,\phi_{\mathbf{i}}(x)\,u_i, \tag{5.5}$$

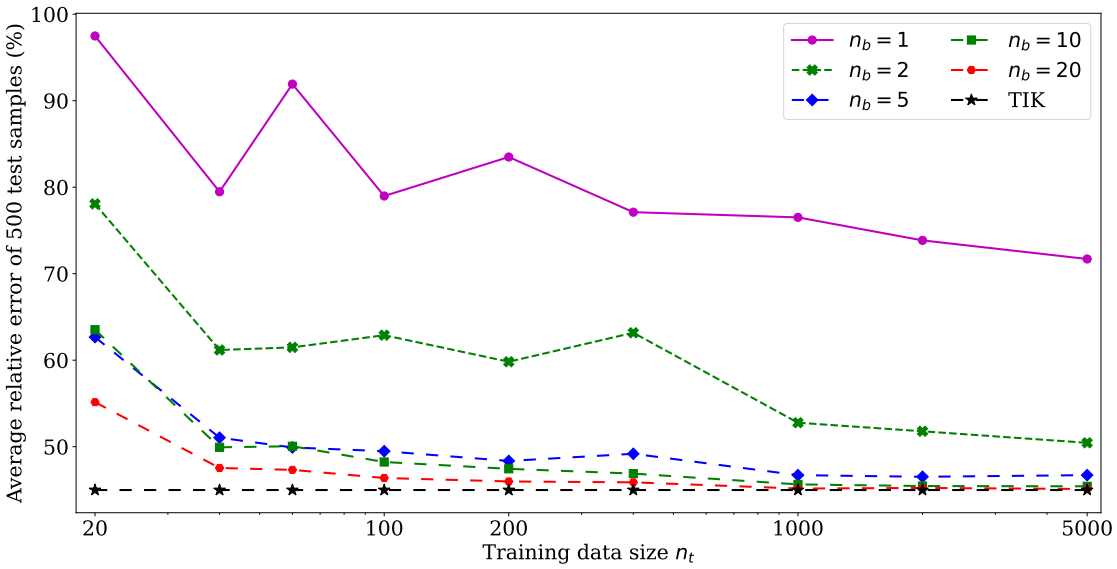

Figure 7: **2D heat conductivity inverse problem.** The average relative error (5.2) as a function of n_b and n_t for `TNet` approach.

where $u_i \sim \mathcal{N}(0,1), i = 1, \ldots, 24$, thus $\boldsymbol{u}_0 = \mathbf{0}, \Gamma = I$, and (λ_i, ϕ_i) are eigenpairs obtained by the eigendecomposition of the covariance operator $7^{\frac{3}{2}}(-\Delta + 49\mathbf{I})^{-2.5}$ with periodic boundary conditions. Next, we discretize an initial vorticity $\omega(x, 0)$, denoted as ω_0, and we solve the Navier-Stokes equation by the stream-function formulation with a pseudospectral method [33] to obtain a discrete representation ω_t of $\omega(x, t)$ at any time t. The observation operator is imposed on solution ω_{10} to form the synthetic observables $\boldsymbol{y}$, then a realization of additive white noise with $\delta = 2\%$ is added to generate a noise-corrupted $\boldsymbol{y}$ sample. A sample of (ω_0, ω_{10}) pair together with the observation points is shown in the middle column of Figure 10.

Similar to the heat conductivity inverse problem in subsection 5.1 and Burgers' equations in subsection SM2.2, we consider two cases of training data. *Case I*: Full data with distinct training samples are used; and *Case II*: we first pick a number of distinct baseline samples n_b smaller than n_t, and then replicate and randomize them to obtain n_t samples for the train data set. We shall compare and contrast results from `nDNN`, `mcDNN`, `TNet`, and Tikhonov solutions.

Case I: Full distinct training samples $n_b = n_t = \{50, 100, 200, 500\}$**.** In Figure 8 are the average relative error (5.2) versus the regularization parameter α over 500 test samples with $n_b = n_t = \{50, 100, 200, 500\}$. The results are shown for `nDNN` (dashed curves), `mcDNN` (dotted curves), `TNet` (colored solids curves), and Tikhonov (TIK: black curve) solutions. The general behavior of the error as a function of regularization parameter is similar to the results for Burgers and heat equations, and thus omitted. Here, *we focus on results at the "best" regularization parameters for all methods.* The optimal regularization parameters of `mcDNN` and `TNet` agree with that of Tikhonov methods, namely $\alpha = 2200$, due to the same reason as explained in the three other numerical problems. Whereas, `nDNN` has small optimal regularization parameters. At the optimal regularization parameter, as summarized in Table 4,

with a given n_t, `TNet` error is smaller than those of `nDNN` and `mcDNN` with twice amount of data. For example, `TNet` solution with $n_t = 100$ incurs an error of 27.14%, smaller than 32.39% and 28.54% of `nDNN` and `mcDNN` with $n_t = 200$. It is not surprising that `TNet` solution tends to converge to Tikhonov (TIK) solution faster as it is designed to do so (see Corollary 4.1) while the others are not. Clearly, without being constrained to the forward map `nDNN` needs the largest amount of data to approximate the inverse map with the same level of accuracy.

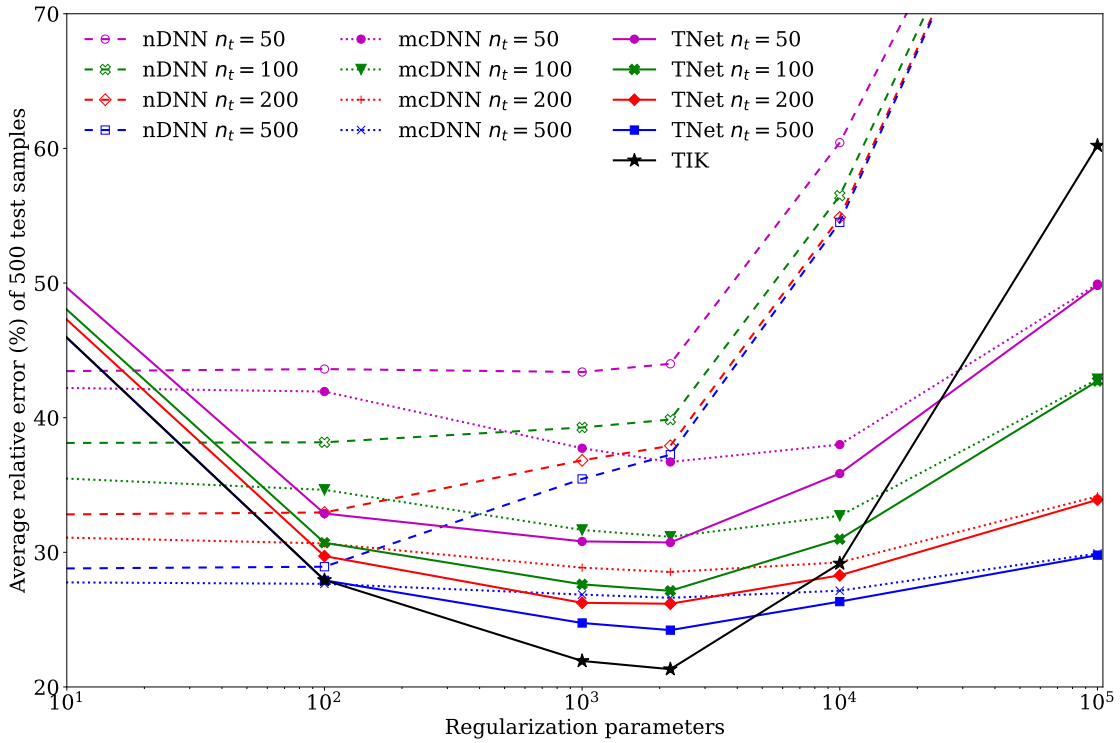

Figure 8: **2D Navier-Stokes equation, Case I.** The average relative error (5.2) versus the regularization parameter α over 500 test samples with $n_b = n_t = \{50, 100, 200, 500\}$. The results are shown for `nDNN` (dashed curves), `mcDNN` (dotted curves), `TNet` (colored solids curves), and Tikhonov (TIK: black curve) solutions.

Case II: Training with $n_b \in \{10, 50\} \leq n_t \in \{50, 100, 200, 500, 1000\}$. Table 5 presents the relative error (5.2) of test data sets obtained by different approaches at the optimal regularization parameters. It can be seen that `nDNN` results are improved as more distinct baseline data pairs are deployed in training data sets. Nevertheless, for any baseline case, replication and randomization to generate n_t data samples, while being more computationally demanding, do not provide additional accuracy in `nDNN` solutions. Again, this implies that the performance of `nDNN` completely relies on the underlying information provided by distinct baseline data. The behavior of `mcDNN` is, on the other hand, not predictable. In particular, replication and randomization improves the results for $n_b = 10$ but not for $n_b = 50$: perhaps due to the data-driven nature, and hence bias, of $\boldsymbol{u}_0^{\texttt{mcDNN}}$ as discussed in section 4 after Corollary 4.1.

Table 4: **2D Navier-Stokes equation, Case I.** The average relative error (5.2) over 500 test samples obtained by `nDNN` ($\alpha \approx 10$), `mcDNN` ($\alpha = 2200$), `TNet` ($\alpha = 2200$) with nested data sets $n_t = 50 \subset n_t = 100 \subset n_t = 200 \subset n_t = 500$, and TIK ($\alpha = 2200$.)

	`nDNN`	`mcDNN`	`TNet`	TIK
$n_t = 50$	43.00	37.86	30.71	21.93
$n_t = 100$	37.98	31.15	27.14	
$n_t = 200$	32.39	28.54	26.18	
$n_t = 500$	28.08	26.62	24.22	

On the contrary, `TNet` results are consistently and significantly improved with replication and randomization for $n_b \in \{10, 50\}$. In particular, for a given n_b, the larger n_t is, the more accurate the `TNet` solution. As further demonstrated in Figure 9, trained with $n_b = 50$ and $n_t = 1000$, the distribution of pointwise relative error for `TNet` is closest to that of Tikhonov solution, and is every where far lower than those of `mcDNN` and `nDNN`. Shown in Figure 10 are the predicted (reconstructed) initial vorticities for an unseen noisy test sample obtained by `nDNN`, `mcDNN`, and `TNet` neural networks for $n_b = 50$ and $n_t = 1000$ along with the Tikhonov reconstruction. Shown in the middle column are the synthetic ground truth initial vorticity and the corresponding final vorticity for reference. Again, `TNet` reconstruction is closest to TIK inversion and `nDNN` provides the most inaccurate solution.

Table 5: **2D Navier-Stokes equation, Case II.** The average relative error (5.2) for `nDNN` ($\alpha \approx 10$), `mcDNN` ($\alpha = 2200$), `TNet`($\alpha = 2200$), and Tikhonov (TIK) ($\alpha = 2200$) over 500-sample test data set obtained with $n_b = \{10, 50\}$ baseline data pairs.

Training data size (n_t)	$n_b = 10$			$n_b = 50$			TIK
	`nDNN`	`mcDNN`	`TNet`	`nDNN`	`mcDNN`	`TNet`	
$n_t = 50$	85.09	73.15	58.71	43.00	37.86	30.71	21.93
$n_t = 100$	85.37	72.58	48.46	42.03	38.80	30.06	
$n_t = 200$	85.87	64.04	39.75	43.17	37.74	29.06	
$n_t = 500$	85.85	50.76	38.28	43.29	37.10	27.91	
$n_t = 1000$	85.52	48.32	34.13	42.85	37.24	26.96	

5.3. Training cost and speedup with deep learning solutions. The training costs for the case of $n_t = 1000$ randomized training samples for heat, Burgers, and Navier-Stokes equations are presented in Table 6. It can be observed that the heat equation requires the least training time, 1.6 hours, while the corresponding times for the Burgers' equation and Navier-Stokes equation are 16 and 20 hours, respectively. It should be noted that executing the forward map and the backpropagation constitutes the majority of the training cost. Table 6 also provides information on the computational cost of reconstructing an unseen test sample using the

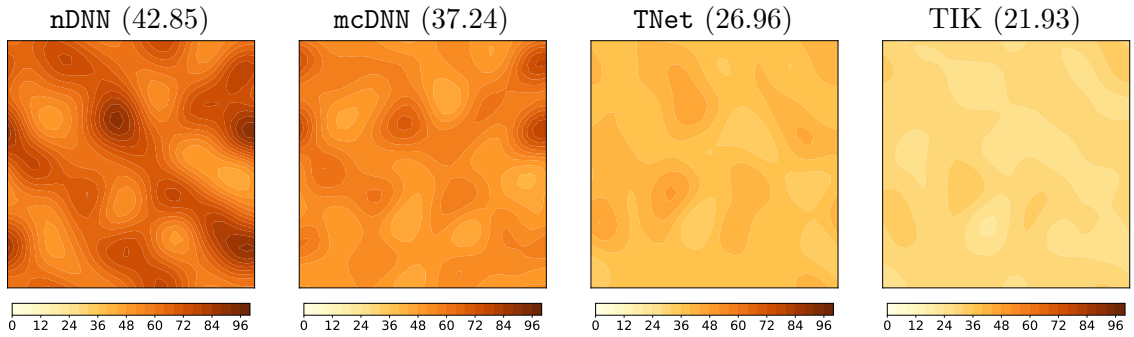

Figure 9: **2D Navier-Stokes equation, Case II.** The distribution of average relative pointwise error (5.1) for `nDNN`, `mcDNN`, `TNet`, and Tikhonov (TIK) over 500 test samples obtained with $n_b = 50$ and $n_t = 1000$. The numbers in the parentheses are the average error (5.2) incurred by these methods.

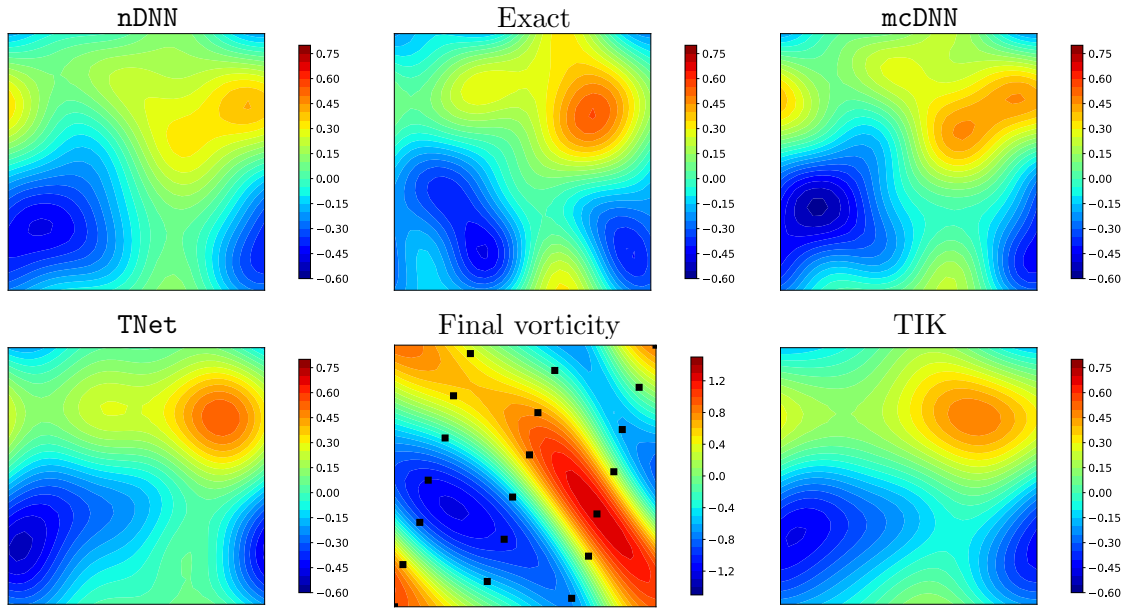

Figure 10: **2D Navier-Stokes equation, Case II.** Predicted (reconstructed) initial vorticities for an unseen noisy test sample obtained by `nDNN`, `mcDNN`, and `TNet` neural networks along with the Tikhonov reconstruction for $n_b = 50$ and $n_t = 1000$. Shown in the middle column are the synthetic ground truth (Exact) initial vorticity and the corresponding final vorticity for reference.

581 classical Tikhonov (TIK) regularization technique and our proposed deep learning approach[2]

[2]Note that the cost for `nDNN` and `mcDNN` is the same as they have the same network architecture.

TNet. We order the first column the increase in complexity from heat equation to Burger's equation to Navier-Stokes equation. Here, the complexity is estimated based on the number of time steps, the operations carried out per time step, and the mesh size. As can be seen, the more complicated the problem is, the more time Tikhonov takes to obtain the solution. Unlike the Tikhonov approach, regardless of the complexity of the problems, the learned TNet inverse map using one hidden layer with 5000 neurons takes the same small amount of time: approximately 0.0003 seconds. Note that the Tikhonov solver is implemented directly in JAX using the default BFGS algorithm with the gradient computed by the default Autograd functionality. Thus, the Tikhonov computation enjoys JAX optimized features including XLA (accelerated linear algebra), JIT (just-in-time compilation), and the nested primitive loop technique. Even with such optimization, Tikhonov is still orders of magnitude slower than TNet. In particular, for the Navier-Stokes equation, TNet is 24,785 times faster than Tikhonov. We expect the computational gain is much more significant for larger-scale 3D time-dependent nonlinear forward problems. Clearly, once trained, obtaining TNet solutions is simply a feed-forward neural network evaluation, which could be close to real-time or real-time depending on the depth and the width of the network.

Table 6: The training cost (measured in hours) for the case of $n_t = 1000$ randomized training samples for all three problems. The computational time (measured in seconds) for inverse solution governed by heat, Burgers, and Navier-Stokes equation using TNet (second column) and Tikhonov (third column) methods, and the speed-up (fourth column) of TNet relative to Tikhonov using NVIDIA A100 GPUs on Lonestar6 at the Texas Advanced Computing Center (TACC).

Inverse Problems	Training (hours)	TNet (seconds)	TIK (seconds)	Speed-up
Heat equation	1.6	2.74×10^{-4}	4.36×10^{-2}	159
Burger's equations	16.0	2.93×10^{-4}	1.08	3,683
Navier-Stokes equation	20.0	2.93×10^{-4}	7.26	24,785

6. Conclusions. We argue that in order for a DNN to generalize well in insufficient data regimes, it should be equipped with information encoded in the underlying mathematical model that is not or partially covered in the data set. In other words, it is natural to require DNN to be aware of the underlying mathematical models (or discretizations) in order for it to be a *reliable and interpretable tool* for sciences and engineering applications. Indeed, we have shown that the proposed model-constrained deep learning approaches are the same as Tikhonov regularization methods for linear inverse problems, while it is not clear if inverse solutions using purely data-driven DL methods are interpretable. We have shown that data randomization can further enhance the robustness and the generalization of our model-constrained deep neural networks. The numerical results not only confirm the theoretical findings but also show that even with as little as 1 training data sample for 1D deconvolution, 5 for inverse heat conductivity, 100 for inverse initial condition for time-dependent 2D Burgers' equation, and 50 for inverse initial condition for 2D Navier-Stokes equations, TNet solutions can be as accurate as Tikhonov solutions while being several orders of magnitude faster. This

is not possible without the model-constrained term, replications, and randomization. Ongoing work is to understand under which conditions `TNet` solution converges to Tikhonov solution for nonlinear inverse problems. Of interest is to estimate the minimum number of distinct baseline training samples to obtain a certain accuracy on average for unseen data. Inversions governed by fluid flows with high Reynolds numbers and shocks are an important class of problems in our portfolio, and they are under investigation. Extension to statistical inversions is also part of our future work. The main limitation of `TNet` is that a differential solver is needed during training. This can be resolved by using a differential numerical library. Another approach is to replace the differential solver with differential residual evaluation. We will report these approaches in future work.

Acknowledgments. This research is partially funded by the National Science Foundation awards NSF-OAC-2212442, NSF-2108320, NSF-1808576 and NSF-CAREER-1845799; by the Department of Energy award DE-SC0018147 and DE-SC0022211. The authors would like to thank Krishnanunni Chandradath Girija, Jau-Uei Chen, Sheroze Sheriffdeen, Jonathan Wittmer, Hwan Goh, and Co Tran for fruitful discussions. The authors also acknowledge the Texas Advanced Computing Center (TACC) at The University of Texas at Austin for providing HPC, visualization, database, or grid resources that have contributed to the research results reported within this paper. URL: http://www.tacc.utexas.edu

REFERENCES

[1] Jonas Adler and Ozan Öktem. Solving ill-posed inverse problems using iterative deep neural networks. *Inverse Problems*, 33(12):124007, 2017.
[2] Hemant K Aggarwal, Merry P Mani, and Mathews Jacob. Modl: Model-based deep learning architecture for inverse problems. *IEEE transactions on medical imaging*, 38(2):394–405, 2018.
[3] O. M. Alifanov. *Inverse Heat Transfer Problems*. Springer Verlag, Berlin, Heidelberg, New-York, 1994.
[4] Kelsey R Allen, Tatiana Lopez-Guevara, Kimberly Stachenfeld, Alvaro Sanchez-Gonzalez, Peter Battaglia, Jessica Hamrick, and Tobias Pfaff. Physical design using differentiable learned simulators. *arXiv preprint arXiv:2202.00728*, 2022.
[5] Guozhong An. The effects of adding noise during backpropagation training on a generalization performance. *Neural computation*, 8(3):643–674, 1996.
[6] Jens Berg and Kaj Nyström. Neural network augmented inverse problems for pdes, 2017.
[7] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag, Berlin, Heidelberg, 2006.
[8] Pavel B Bochev and Max D Gunzburger. *Least-squares finite element methods*. Springer, 2006.
[9] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.
[10] S. C. Brenner and L. R. Scott. *The Mathematical Theory of Finite Element Methods*. Springer Verlag, Berlin, Heidelberg, New York, second edition, 2002.
[11] Tan Bui-Thanh. A unified and constructive framework for the universality of neural networks, 2021.
[12] Tan Bui-Thanh, Carsten Burstedde, Omar Ghattas, James Martin, Georg Stadler, and Lucas C. Wilcox. Extreme-scale UQ for Bayesian inverse problems governed by PDEs. In *SC12: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2012. Gordon Bell Prize finalist, http://users.ices.utexas.edu/%7Etanbui/PublishedPapers/sc12.pdf.
[13] Yuyao Chen, Lu Lu, George Em Karniadakis, and Luca Dal Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Opt. Express*, 28(8):11618–11633, Apr 2020.
[14] P. G. Ciarlet. *The finite element method for elliptic problems*, volume 40 of *Classics in Applied Mathemat-*

ics. SIAM (SIAM), Philadelphia, PA, 2002. Reprint of the 1978 original [North-Holland, Amsterdam; MR0520174 (58 #25001)].

[15] Paul G Constantine, Carson Kent, and Tan Bui-Thanh. Accelerating markov chain monte carlo with active subspaces. *SIAM Journal on Scientific Computing*, 38(5):A2779–A2805, 2016.

[16] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals and Systems*, 2(4):303–314, Dec 1989.

[17] Alexandre Ern and Jean-Luc Guermond. *Theory and Practice of Finite Elements*, volume 159 of *Applied Mathematical Sciences*. Spinger-Verlag, 2004.

[18] Tiffany Fan, Kailai Xu, Jay Pathak, and Eric Darve. Solving inverse problems in steady-state navier-stokes equations using deep neural networks. *arXiv preprint arXiv:2008.13074*, 2020.

[19] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In Yee Whye Teh and Mike Titterington, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.

[20] Hwan Goh, Sheroze Sheriffdeen, and Tan Bui-Thanh. Solving forward and inverse problems using autoencoders, 2019.

[21] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[22] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

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

[24] Jiaqi Jiang, Mingkun Chen, and Jonathan A. Fan. Deep neural networks for the evaluation and design of photonic devices. *Nature Reviews Materials*, Dec 2020.

[25] Yuchen Jin, Qiuyang Shen, Xuqing Wu, Jiefu Chen, and Yueqin Huang. A physics-driven deep-learning network for solving nonlinear inverse problems. *Petrophysics-The SPWLA Journal of Formation Evaluation and Reservoir Description*, 61(01):86–98, 2020.

[26] Jesse Johnson. Deep, skinny neural networks are not universal approximators. In *International Conference on Learning Representations*, 2019.

[27] Jari Kaipio and Erkki Somersalo. *Statistical and Computational Inverse Problems*, volume 160 of *Applied Mathematical Sciences*. Springer-Verlag, New York, 2005.

[28] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[29] Keisuke Kojima, Bingnan Wang, Ulugbek Kamilov, Toshiaki Koike-Akino, and Kieran Parsons. Acceleration of fdtd-based inverse design using a neural network approach. In *Advanced Photonics 2017 (IPR, NOMA, Sensors, Networks, SPPCom, PS)*, page ITu1A.4. Optical Society of America, 2017.

[30] Dimitri Komatitsch, Jeroen Ritsema, and Jeroen Tromp. The spectral-element method, Beowulf computing, and global seismology. *Science*, 298:1737–1742, 2002.

[31] Matthieu Lefebvre, Ebru Bozda, Henri Calandra, Judy Hill, Wenjie Lei, Daniel Peter, Norbert Podhorszki, David Pugmire, Herurisa Rusmanugroho, James Smith, and Jeroen Tromp. A data centric view of large-scale seismic imaging workflows. *Supercomputing (SC) 13*, 2013. Invited paper.

[32] Housen Li, Johannes Schwab, Stephan Antholzer, and Markus Haltmeier. Nett: Solving inverse problems with deep neural networks. *Inverse Problems*, 36(6):065005, 2020.

[33] Zongyi Li, Nikola Kovachki, Kamyar Azizzadenesheli, Burigede Liu, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Fourier neural operator for parametric partial differential equations, 2020.

[34] Dianjing Liu, Yixuan Tan, Erfan Khoram, and Zongfu Yu. Training deep neural networks for the inverse design of nanophotonic structures. *ACS Photonics*, 5(4):1365–1369, 2018.

[35] Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021.

[36] Lu Lu, Raphael Pestourie, Wenjie Yao, Zhicheng Wang, Francesc Verdugo, and Steven G. Johnson. Physics-informed neural networks with hard constraints for inverse design, 2021.

[37] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The expressive power of neural networks: A view from the width. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fer-

gus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[38] Sebastian Lunz, Ozan Öktem, and Carola-Bibiane Schönlieb. Adversarial regularizers in inverse problems. *Advances in neural information processing systems*, 31, 2018.

[39] Jie Luo, Xun Li, Xinyuan Zhang, Jiajie Guo, Wei Liu, Yun Lai, Yaohui Zhan, and Min Huang. Deep-learning-enabled inverse engineering of multi-wavelength invisibility-to-superscattering switching with phase-change materials. *Opt. Express*, 29(7):10527–10537, Mar 2021.

[40] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. *Foundations of Machine Learning*. The MIT Press, 2012.

[41] Jennifer L Mueller and Samuli Siltanen. *Linear and nonlinear inverse problems with practical applications*. SIAM, 2012.

[42] Hai V. Nguyen and Tan Bui-Thanh. Model-constrained deep learning approaches for inverse problems, 2021.

[43] Jorge Nocedal and Stephen J Wright. *Numerical optimization*. Springer, 1999.

[44] Dean S. Oliver, Albert C. Reynolds, and Ning Liu. *Inverse theory for petroleum reservoir characterization and history matching*. Cambidge University Press, 2008.

[45] Samira Pakravan, Pouria A Mistani, Miguel A Aragon-Calvo, and Frederic Gibou. Solving inverse-pde problems with physics-aware neural networks. *Journal of Computational Physics*, 440:110414, 2021.

[46] Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[47] Raphaël Pestourie, Youssef Mroueh, Thanh V. Nguyen, Payel Das, and Steven G. Johnson. Active learning of deep surrogates for pdes: application to metasurface design. *npj Computational Materials*, 6(1):164, Oct 2020.

[48] John Peurifoy, Yichen Shen, Li Jing, Yi Yang, Fidel Cano-Renteria, Brendan G. DeLacy, John D. Joannopoulos, Max Tegmark, and Marin Soljačić. Nanophotonic particle simulation and inverse design using artificial neural networks. *Science Advances*, 4(6), 2018.

[49] Tobias Pfaff, Meire Fortunato, Alvaro Sanchez-Gonzalez, and Peter W Battaglia. Learning mesh-based simulation with graph networks. *arXiv preprint arXiv:2010.03409*, 2020.

[50] 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, 2019.

[51] 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, 2019.

[52] Maziar Raissi and George Em Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *Journal of Computational Physics*, 357:125 – 141, 2018.

[53] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Machine learning of linear differential equations using gaussian processes. *Journal of Computational Physics*, 348:683 – 693, 2017.

[54] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. *arXiv preprint arXiv:1711.10566*, 2017.

[55] Shai Shalev-Shwartz and Shai Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, USA, 2014.

[56] Anand Pratap Singh, Shivaji Medida, and Karthik Duraisamy. Machine-learning-augmented predictive modeling of turbulent separated flows over airfoils. *AIAA journal*, 55(7):2215–2227, 2017.

[57] Sunae So, Trevon Badloe, Jaebum Noh, Jorge Bravo-Abad, and Junsuk Rho. Deep learning enabled inverse design in nanophotonics. *Nanophotonics*, 9(5):474, February 2020.

[58] Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. On the importance of initialization and momentum in deep learning. In Sanjoy Dasgupta and David McAllester, editors, *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pages 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

[59] Mohammad H. Tahersima, Keisuke Kojima, Toshiaki Koike-Akino, Devesh Jha, Bingnan Wang, Chungwei Lin, and Kieran Parsons. Deep neural network inverse design of integrated photonic power splitters. *Scientific Reports*, 9(1):1368, Feb 2019.

[60] Albert Tarantola. *Inverse Problem Theory and Methods for Model Parameter Estimation*. SIAM, Philadelphia, PA, 2005.

[61] Andrei Nikolaevich Tikhonov, AV Goncharsky, VV Stepanov, and Anatoly G Yagola. *Numerical methods for the solution of ill-posed problems*, volume 328. Springer Science & Business Media, 1995.

[62] Rohit K. Tripathy and Ilias Bilionis. Deep uq: Learning deep neural network surrogate models for high dimensional uncertainty quantification. *Journal of Computational Physics*, 375:565 – 588, 2018.

[63] Curtis R Vogel. *Computational methods for inverse problems*. SIAM, 2002.

[64] Daniel A. White, William J. Arrighi, Jun Kudo, and Seth E. Watts. Multiscale topology optimization using neural network surrogate models. *Computer Methods in Applied Mechanics and Engineering*, 346:1118–1135, 2019.

[65] Yue Wu, Youzuo Lin, and Zheng Zhou. Inversionnet: Accurate and efficient seismic waveform inversion with convolutional neural networks. In *2018 SEG International Exposition and Annual Meeting*. OnePetro, 2018.

[66] Yibo Yang and Paris Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 2019.

[67] Qingqing Zhao, David B Lindell, and Gordon Wetzstein. Learning to solve pde-constrained inverse problems with graph networks. *arXiv preprint arXiv:2206.00711*, 2022.