VOLUME 1, ISSUE 1 **2020**

JOURNAL OF MACHINE LEARNING FOR MODELING AND COMPUTING

DONGBIN XIU
Department of Mathematics
The Ohio State University
Columbus, Ohio 43210, USA

begell
New York • Connecticut

AIMS AND SCOPE

The *Journal of Machine Learning for Modeling and Computing* (JMLMC) focuses on the study of machine learning methods for modeling and scientific computing. The scope of the journal includes, but is not limited to, research of the following types: (1) the use of machine learning techniques to model real-world problems such as physical systems, social sciences, biology, etc.; (2) the development of novel numerical strategies, in conjunction of machine learning methods, to facilitate practical computation; and (3) the fundamental mathematical and numerical analysis for understanding machine learning methods.

Journal of Machine Learning for Modeling and Computing (ISSN 2689-3967) is published quarterly and owned by Begell House, Inc., 50 North Street, Danbury, CT 06810, telephone (203) 456-6161. USA subscription rate for 2020 is $800.00. Add $10.00 per issue for foreign airmail shipping and handling fees to all orders shipped outside the United States or Canada. Subscriptions are payable in advance. Subscriptions are entered on an annual basis, i.e., January to December. For immediate service and charge card sales, call (203) 456-6161 Monday through Friday 9 AM–5 PM EST. Fax orders to (203) 456-6167. Send written orders to Subscriptions Department, Begell House, Inc., 50 North Street, Danbury, CT 06810. You can also visit our website at http://www.begellhouse.com/ or http://www.dl.begellhouse.com/.

Printed July 23, 2020

JOURNAL OF MACHINE LEARNING FOR MODELING AND COMPUTING

JOURNAL OF MACHINE LEARNING FOR MODELING AND COMPUTING

Volume 1, Issue 1 **2020**

TABLE OF CONTENTS

TENSOR BASIS GAUSSIAN PROCESS MODELS OF HYPERELASTIC MATERIALS

Ari L. Frankel, Reese E. Jones, & Laura P. Swiler*

Sandia National Laboratories, Livermore, California, 94551, USA

*Address all correspondence to: Ari L. Frankel, Sandia National Laboratories, Quantitative Modeling and Analysis, Livermore, CA 94551, USA, E-mail: alfrank@sandia.gov

Original Manuscript Submitted: 12/18/2019; Final Draft Received: 5/5/2020

In this work, we develop Gaussian process regression (GPR) models of isotropic hyperelastic material behavior. First, we consider the direct approach of modeling the components of the Cauchy stress tensor as a function of the components of the Finger stretch tensor in a Gaussian process. We then consider an improvement on this approach that embeds rotational invariance of the stress-stretch constitutive relation in the GPR representation. This approach requires fewer training examples and achieves higher accuracy while maintaining invariance to rotations exactly. Finally, we consider an approach that recovers the strain-energy density function and derives the stress tensor from this potential. Although the error of this model for predicting the stress tensor is higher, the strain-energy density is recovered with high accuracy from limited training data. The approaches presented here are examples of physics-informed machine learning. They go beyond purely data-driven approaches by embedding the physical system constraints directly into the Gaussian process representation of materials models.

KEY WORDS: *Gaussian process regression, hyperelastic materials, physics-informed machine learning*

1. INTRODUCTION

Machine learning models have seen an explosion in development and application in recent years due to their flexibility and capacity for capturing the trends in complex systems (Hastie et al., 2016). Provided with sufficient data, the parameters of the model may be calibrated in such a way that the model gives high fidelity representations of the underlying data generating process (Raissi et al., 2017; Jones et al., 2018; Frankel et al., 2019a,b). Moreover, computational capabilities have grown such that constructing deep learning models over datasets of tens of thousands to millions of data points is now feasible (Dean et al., 2012). There remain, however, many applications in which the amount of data present is insufficient on its own to properly train the machine learning model. This may be due to a prohibitively large model that requires a correspondingly large amount of data to train and where training data is expensive to acquire. Furthermore, even with a wealth of data, it is possible that the machine learning model may yield behavior that is inconsistent with the expected trend of the model when the model is queried in an extrapolatory regime.

In such cases it is appealing to turn to a framework that allows the incorporation of physical principles and other *a priori* information to supplement the limited data and regularize the behavior of the model. This information can be as simple as a known set of constraints that the

2689–3967/20/$35.00

regressor must satisfy, such as positivity or monotonicity with respect to a particular variable, or can be as complex as knowledge of the underlying data-generating process in the form of a partial differential equation. Consequently, the past few years have seen great interest in "physics-constrained" machine learning algorithms within the scientific computing community (Brunton et al., 2016; Jones et al., 2018; Lee and Carlberg, 2020; Ling et al., 2016; Lusch et al., 2018; Pan and Duraisamy, 2018; Raissi and Karniadakis, 2018). The overview paper by Karpatne et al. (2017) provides a taxonomy for theory-guided data science, with the goal of incorporating scientific consistency in the learning of generalizable models. Much research in physics-informed machine learning has focused on incorporating constraints in neural networks (Ling et al., 2016; Jones et al., 2018), often through the use of objective/loss functions that penalize constraint violation (Magiera et al., 2020).

In contrast, the focus in this paper is to incorporate rotational symmetries directly and exactly into Gaussian process (GP) representations of physical response functions. This approach has the advantages of avoiding the burden of a large training set that comes with a neural network model, and the inexact satisfaction of constraints that come with penalization of constraints in the loss function. There has been significant interest in the incorporation of constraints into Gaussian process regression (GPR) models recently (Bachoc et al., 2019; Da Veiga and Marrel, 2012; Jensen et al., 2013; López-Lopera et al., 2018; Raissi et al., 2017; Riihimäki and Vehtari, 2010; Solak et al., 2003; Yang et al., 2018). Many of these approaches leverage the analytic formulation of the GP to incorporate constraints through the likelihood function or i.e. the covariance function.

In this paper, the task of learning the six components of a symmetric stress tensor from the six components of a symmetric stretch tensor is formulated through a series of transformations so that it becomes a regression task of learning three coefficients that are a function of three invariants of the problem. The main contribution of this paper is the extension of GPR to enforce rotational invariance through a tensor basis expansion.

The paper is organized as follows: Section 2 presents an overview of constitutive models for hyperelastic materials. Sections 3 and 4 present GPR and the extension to a tensor basis GP, respectively. Section 5 presents a further extension of the tensor-basis GP to handle the strain energy potential. Section 6 provides results for a particular hyperelastic Mooney–Rivlin material, and Section 7 provides concluding discussion.

2. HYPERELASTIC MATERIALS

A hyperelastic material is a material that remains elastic (nondissipative) in the finite/large strain regime. In this context the fundamental deformation measure is the 3×3 deformation gradient tensor $\mathbf{F}$:

$$\mathbf{F} = \frac{\partial \mathbf{x}}{\partial \mathbf{X}}, \tag{1}$$

which is the derivative of the current position $\mathbf{x}$ with respect to position $\mathbf{X}$ of the same material point in a chosen reference configuration. In an Eulerian frame (in an inertial frame from which the tensors are measured), the Finger tensor $\mathbf{B}$,

$$\mathbf{B} = \mathbf{F}\mathbf{F}^T, \tag{2}$$

is the typical finite stretch measure, which is directly related to the Almansi strain, which measures the total deformation that a material has undergone relative to its initial configuration. Note

that this tensor is symmetric positive definite, as its eigenvalues are equal to the relative changes in length along principal axes, and negative lengths are not possible. [The choice of the Finger tensor is not limiting in terms of the generality of this formulation, given the equivalence of strain measures provided by the Seth–Hill (Hill, 1968) and Doyle–Ericksen (Doyle and Ericksen, 1956) formulae.] The deformation of a hyperelastic material requires an applied stress state, associated with a certain amount of energy, to arrive at that deformed state. For a hyperelastic material, the stress is solely a function of the current stretch (or strain) of the material. Hence, the major goal of material modeling of hyperelastic materials is to construct constitutive relations between the kinematic variable $\mathbf{B}$ and the corresponding dynamic variable, the 3×3 Cauchy stress tensor

$$\boldsymbol{\sigma} = \mathbf{f}(\mathbf{B}) = \frac{2}{|\mathbf{B}|^{1/2}} \mathbf{B} \frac{\partial \Phi}{\partial \mathbf{B}}, \tag{3}$$

which for a hyperelastic material is given by the derivative with respect to all the tensor components of a potential, namely the strain energy density Φ, and $|\mathbf{B}|$ is the determinant of the Finger tensor. Without additional arguments such as a structure tensor, this formulation is appropriate for isotropic hyperelastic materials. For further details please consult Malvern (1969), Ogden (1997), and Gurtin (1982).

Typical approaches to model these relations seek semiempirical formulations for the strain energy density with some parameters to be fit, which are then fit to experimental data. An example of this type of formulation will be discussed in a later section. In this work we consider nonparametric modeling of hyperelastic material responses.

3. GAUSSIAN PROCESS REGRESSION

GPR provides a nonparametric model for a response function given an input set of training data through a Bayesian update involving an assumed prior distribution and a likelihood tying the posterior distribution to observed data. We denote a GP prior for a function f by

$$f \sim \mathcal{GP}(0, K), \tag{4}$$

where we assume the GP has a nominal mean of 0, without loss of generality, and is described by a covariance function K. We adopt the commonly employed squared-exponential covariance function:

$$K(x, x') = \theta_1 \exp\left(-\theta_2 |x - x'|^2\right), \tag{5}$$

which has a scale parameter θ_1 and a (inverse-square) length parameter θ_2.

A GP is defined such that any finite collection of realizations from the process are governed by a multivariate normal distribution. That is, for any set of observed realizations X and prediction points X^* with corresponding function values $f(\mathsf{X})$ and $f(\mathsf{X}^*)$, the probability distribution $p(f(\mathsf{X}^*), f(\mathsf{X})|\mathsf{X}^*, \mathsf{X})$ is given by

$$\begin{bmatrix} f(\mathsf{X}) \\ f(\mathsf{X}^*) \end{bmatrix} = \mathcal{N}\left(\begin{bmatrix} 0 \\ 0 \end{bmatrix}, \begin{bmatrix} K(\mathsf{X}, \mathsf{X}) & K(\mathsf{X}, \mathsf{X}^*) \\ K(\mathsf{X}^*, \mathsf{X}) & K(\mathsf{X}^*, \mathsf{X}^*) \end{bmatrix} \right), \tag{6}$$

where it is understood that the vectors and matrices presented are given in block form for multiple instances in X and X^*. GPR uses a GP as a prior over the function space for the data-generating process, and predictions proceed through the use of Bayes' rule. Upon observation of some initial set of noisy data points $\mathsf{y} = y(\mathsf{X}) \sim \mathcal{N}(f(\mathsf{X}), \varepsilon^2)$, where ε^2 is the variance of

Gaussian noise in the function values, the probability distribution of the values of the GP at X may be determined by forming the (posterior) conditional distribution of $p(\mathsf{f}|\mathsf{y},\mathsf{X})$:

$$p(\mathsf{f}|\mathsf{y},\mathsf{X}) = \frac{p(\mathsf{y}|\mathsf{f})p(\mathsf{f}|\mathsf{X})}{p(\mathsf{y}|\mathsf{X})} = \frac{p(\mathsf{y}|\mathsf{f})p(\mathsf{f}|\mathsf{X})}{\int p(\mathsf{y}|\mathsf{f})p(\mathsf{f}|\mathsf{X})\mathrm{d}\mathsf{f}}, \tag{7}$$

where $\mathsf{f} = f(\mathsf{X})$ and the Gaussian likelihood is given by

$$p(\mathsf{y}|\mathsf{f}) = \prod_{i=1}^{N} \frac{1}{\sqrt{2\pi\varepsilon^2}} \exp\left[-\frac{(y_i - f_i)^2}{2\varepsilon^2}\right]. \tag{8}$$

The push-forward posterior distribution for predictions at a new set of points X^* has the following analytical solution:

$$\mathsf{f}^* = f(\mathsf{X}^*)|\mathsf{X},\mathsf{y},\mathsf{X}^* \sim \mathcal{N}\left[\mathsf{K}^*(\mathsf{K}+\varepsilon^2\mathsf{I})^{-1}\mathsf{y},\ \mathsf{K}^{**} - \mathsf{K}^*(\mathsf{K}+\varepsilon^2\mathsf{I})^{-1}\mathsf{K}^{*T}\right], \tag{9}$$

where I denotes the identity matrix, $\mathsf{K} = K(\mathsf{X},\mathsf{X})$, $\mathsf{K}^* = K(\mathsf{X}^*,\mathsf{X})$, and $\mathsf{K}^{**} = K(\mathsf{X}^*,\mathsf{X}^*)$. Then the predictive mean of the distribution at any new points X^* is given by

$$\mathbb{E}[\mathsf{y}^*|\mathsf{X},\mathsf{y},\mathsf{X}^*] = \mathsf{K}^*(\mathsf{K}+\varepsilon^2\mathsf{I})^{-1}\mathsf{y}, \tag{10}$$

and the predictive variance, assuming the same noise level, is given by

$$\mathbb{V}[\mathsf{y}^*|\mathsf{X},\mathsf{y},\mathsf{X}^*] = \mathsf{K}^{**} - \mathsf{K}^*(\mathsf{K}+\varepsilon^2\mathsf{I})^{-1}\mathsf{K}^{*T} + \varepsilon^2\mathsf{I}, \tag{11}$$

where $\mathsf{y}^* = y(\mathsf{X}^*)$. This result shows the combination of uncertainty in the prediction due to epistemic uncertainty in the mean process (the first two terms on the right side) plus the aleatoric uncertainty of inherent variability in the measurements (the last term on the right side). In this work, although we will work with noiseless data, we assume a value of $\varepsilon^2 = 10^{-10}$ in order to regularize the inversion of the covariance matrix.

The task that dominates the computational expense in constructing this model is the inversion of $(\mathsf{K}+\varepsilon^2\mathsf{I})$, or, equivalently, the solution of the linear system based on $(\mathsf{K}+\varepsilon^2\mathsf{I})$ for either the mean or variance evaluations. Since K is dense, the scaling is typically $\mathcal{O}(N^3)$ for N training points. This limitation constrains GPR to at most $N = 10{,}000$ since memory and computation time become prohibitive, although for this work we found that $N = \mathcal{O}(100)$ was sufficient to reach reasonable accuracy, as will be shown later. Nominally the matrix is symmetric positive semidefinite, which enables efficient solution by Cholesky decompositions, although ill-conditioning is frequently an issue, especially for large dataset sizes. Ill-conditioning requires adding a nugget or large noise ($\varepsilon \gg 1$) term to the covariance matrix to regularize the solution, using pseudoinverses via the singular value decomposition of the covariance matrix, or other greedy subset selection to reduce the matrix size.

It appears at first glance that the variance in Eq. (11) is nominally independent of the actual point values y, and only depends on the locations of the selected data points X. This is true for a fixed covariance function; however, we are typically interested in changing the GP hyperparameters to maximize the accuracy of the GP while balancing the model complexity. Traditionally, this is managed by tuning the hyperparameters to optimize $p(\mathsf{y}|\mathsf{X})$, which is the marginal-likelihood of the GP, and is frequently called the "model evidence." Equivalently, we may optimize the logarithm of the model evidence $L = \log p(\mathsf{y}|\mathsf{X})$ for numerical stability reasons:

$$L = -\frac{1}{2}\mathsf{y}^T\left(\mathsf{K}+\varepsilon^2\mathsf{I}\right)^{-1}\mathsf{y} - \frac{1}{2}\log|\mathsf{K}+\varepsilon^2\mathsf{I}| - \frac{N}{2}\log 2\pi. \tag{12}$$

That is, we choose to tune the covariance hyperparameters θ_1 and θ_2 in Eq. (5) in order to maximize L. It is worth noting that solving this optimization problem requires multiple inversions of a dense covariance matrix, further increasing the cost of performing the inference. Further discussion of this approach can be found in Rasmussen and Williams (2006).

4. TENSOR BASIS GAUSSIAN PROCESS

In this section we show how the standard GPR described in the previous section may be adapted to enforce rotational invariance through a tensor basis expansion. We call this formulation a tensor basis Gaussian process (TBGP).

4.1 Tensor Basis Expansion

We consider the generic hyperelastic constitutive model of the form

$$\boldsymbol{\sigma} = \mathbf{f}(\mathbf{B}), \tag{13}$$

which, for any given analytic tensor valued function $\mathbf{f}$, may be expanded in an infinite series in terms of $\mathbf{B}$ with fixed coefficients $\bar{c}_n$:

$$\boldsymbol{\sigma} = \sum_{n=0}^{\infty} \bar{c}_n \mathbf{B}^n. \tag{14}$$

Note that $\mathbf{B}$ is symmetric, positive definite, and hence has a complete eigenbasis. Furthermore, it is clear that $\boldsymbol{\sigma}$ and $\mathbf{B}$ are coaxial, (i.e., have the same eigenbasis), due to the fact that we are considering an isotropic material. Since the tensors of interest are symmetric and of size 3×3, the Cayley-Hamilton theorem states that the tensor $\mathbf{B}$ satisfies its corresponding characteristic polynomial

$$\mathbf{B}^3 - I_1\mathbf{B}^2 + I_2\mathbf{B} - I_3\mathbf{I} = 0, \tag{15}$$

where we have defined the tensor invariants

$$\begin{aligned} I_1 &= \mathrm{tr}(\mathbf{B}) &&= \lambda_{B_1} + \lambda_{B_2} + \lambda_{B_3}, \\ I_2 &= \frac{1}{2}\left[\mathrm{tr}(\mathbf{B})^2 - \mathrm{tr}(\mathbf{B}^2)\right] &&= \lambda_{B_1}\lambda_{B_2} + \lambda_{B_2}\lambda_{B_3} + \lambda_{B_3}\lambda_{B_1}, \\ I_3 &= \det(\mathbf{B}) &&= \lambda_{B_1}\lambda_{B_2}\lambda_{B_3}, \end{aligned} \tag{16}$$

so-called because they are invariant under similarity transformations (i.e., rotations) of $\mathbf{B}$. Here, λ_{B_1}, λ_{B_2}, and λ_{B_3} are the eigenvalues of $\mathbf{B}$, which are also a complete set of invariants. The theorem can be used as a recursion relation to write all powers of $\mathbf{B}$ higher than 2 in terms of $\mathbf{I}$, $\mathbf{B}$, and $\mathbf{B}^2$ with coefficients that depend on the invariants and the unknown $\bar{c}_i$. Rather than seeking to identify the infinite number of fixed coefficients $\bar{c}_i$ for a given constitutive relation, our task reduces to finding the three coefficients in the series expansion

$$\boldsymbol{\sigma} = c_0(I_1, I_2, I_3)\mathbf{I} + c_1(I_1, I_2, I_3)\mathbf{B} + c_2(I_1, I_2, I_3)\mathbf{B}^2, \tag{17}$$

where c_i is a function of the invariants; this notation will be suppressed for the remainder of this work for clarity.

One issue in many machine learning models of constitutive relations is that they do not always preserve rotational objectivity; that is, rotating the frame in which the deformations are measured should not change the physics predictions, but the machine learning models do not preserve this. For example, the GP formulation in Section 2 does not ensure that rotating the basis for $\mathbf{B}$ (while keeping the components fixed) would give the corresponding fixed components for the stress predictions in the rotated basis. The advantage of this reduced expansion is that it embeds rotational objectivity in its structure. To see this, let $\mathbf{R}$ be an orthogonal/rotation tensor with inverse given by $\mathbf{R}^{-1} = \mathbf{R}^T$. The rotation of $\boldsymbol{\sigma}$ in the original coordinate frame to the frame defined by $\mathbf{R}$ is given by

$$\boldsymbol{\sigma}' \equiv \mathbf{R}\boldsymbol{\sigma}\mathbf{R}^T = \mathbf{R}\mathbf{f}(\mathbf{B})\mathbf{R}^T. \tag{18}$$

Invoking the tensor basis expansion gives

$$\begin{aligned} \boldsymbol{\sigma}' &= c_0\mathbf{R}\mathbf{I}\mathbf{R}^T + c_1\mathbf{R}\mathbf{B}\mathbf{R}^T + c_2\mathbf{R}\mathbf{B}^2\mathbf{R}^T \\ &= c_0\mathbf{R}\mathbf{R}^T + c_1\mathbf{R}\mathbf{B}\mathbf{R}^T + c_2\mathbf{R}\mathbf{B}\mathbf{R}^T\mathbf{R}\mathbf{B}\mathbf{R}^T = \mathbf{f}(\mathbf{R}\mathbf{B}\mathbf{R}^T) \equiv \mathbf{f}(\mathbf{B}'), \end{aligned} \tag{19}$$

which holds since the eigenvalues, and hence invariants and the coefficient functions, do not change upon application of $\mathbf{R}$. In general, an Eulerian tensor function of an Eulerian tensor argument must be objective in the sense it responds to a rotation of its argument with a corresponding rotation of the function value:

$$\mathbf{R}\mathbf{f}(\mathbf{B})\mathbf{R}^T = \mathbf{f}(\mathbf{R}\mathbf{B}\mathbf{R}^T). \tag{20}$$

This is precisely what we refer to as rotational invariance.

4.2 Application to Gaussian Process Modeling

The task of regression now falls to learning the coefficients c_0, c_1, and c_2 as a function of the invariants. This task is compressed from the original problem of having to learn six stress components from six strain components with the added benefit of enforcing the rotational invariance that provides this reduction. Since GPR guarantees theoretical asymptotic consistency of the predictions, using GPR to learn the coefficients as a function of the invariants indirectly guarantees asymptotic consistency of the stress tensor predictions since they are just linear combinations of the three coefficients.

Suppose we are given a dataset of pairs of tensors ($\mathbf{B}$, $\boldsymbol{\sigma}$). Under the assumption that the tensors may be diagonalized, the collinearity of the tensor basis expansion [Eq. (17)] implies that they are diagonalized by the same eigenvector matrix $\mathbf{Q}$ but with different eigenvalue matrices, $\boldsymbol{\Lambda}_{\boldsymbol{\sigma}}$ and $\boldsymbol{\Lambda}_{\mathbf{B}}$, respectively:

$$\boldsymbol{\sigma} = \mathbf{Q}\boldsymbol{\Lambda}_{\boldsymbol{\sigma}}\mathbf{Q}^T, \tag{21}$$

$$\mathbf{B} = \mathbf{Q}\boldsymbol{\Lambda}_{\mathbf{B}}\mathbf{Q}^T. \tag{22}$$

Then for the given input-output pair, the values of the coefficients for the given set of eigenvalues are given by the solution to a 3×3 linear system of equations as

$$\begin{bmatrix} \lambda_{\sigma_1} \\ \lambda_{\sigma_2} \\ \lambda_{\sigma_3} \end{bmatrix} = \begin{bmatrix} 1 & \lambda_{B_1} & \lambda_{B_1}^2 \\ 1 & \lambda_{B_2} & \lambda_{B_2}^2 \\ 1 & \lambda_{B_3} & \lambda_{B_3}^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}. \tag{23}$$

This linear system may be inverted easily to yield the coefficients $c_i(\mathbf{B}, \boldsymbol{\sigma})$ for each training point in the dataset, and the invariants I_i of $\mathbf{B}$ may also be computed straightforwardly from the eigenvalues given in Eq. (16). The three invariants I_i for each observation are accumulated into a matrix. They take the place of the feature matrix X presented in the previous section, and a corresponding matrix of c_i replaces f. We can thus readily extend the GPR approach to infer the coefficients as functions of the invariants of the input tensors and thus construct the representation of the function $\boldsymbol{\sigma} = c_0\mathbf{I} + c_1\mathbf{B} + c_2\mathbf{B}^2$ given in Eq. (17).

In cases where $\mathbf{B}$ has repeated eigenvalues, Eq. (23), as it stands, is not invertible. Physically, this corresponds to the case where equal deformations are applied along different axes. In such a case, the Cayley-Hamilton theorem and continuity of the deformation to stress mapping require that the derivative of the stress eigenvalue with respect to the repeated Finger tensor eigenvalue must be zero. If $\lambda_{B_1} = \lambda_{B_2} \neq \lambda_{B_3}$,

$$0 = c_1 + 2\lambda_{B_1} c_2, \tag{24}$$

should be substituted for one of the redundant equations. If $\lambda_{B_1} = \lambda_{B_2} = \lambda_{B_3}$, the second derivative must also be zero, giving the additional equation

$$c_2 = 0, \tag{25}$$

to replace another of the redundant equations in the system of equations, Eq. (23), in which case the result is trivially $c_0 = \lambda_{\sigma_1}$ and $c_1 = c_2 = 0$. This can be interpreted physically as a volumetric deformation leads to a pressure for an isotropic, elastic material, as expected. Combining these auxiliary conditions with the fact that $\mathbf{B}$ is symmetric positive definite, it is always possible to map the Finger deformation and Cauchy stress tensors to set of coefficients that satisfy the tensor basis expansion and forms a well-posed regression.

One issue with this approach is that the inclusion of measurement error of the stress tensor is not straightforward. Including Gaussian errors in the coefficients will lead to a more complex noise process in the stress predictions, where the stress tensor components will have unequal variances and potentially be correlated. While in this work we are considering noiseless data, extending GPR to consider noisy data is a problem worth considering in future work.

4.3 Example: Matrix Exponential

To illustrate the impact of embedding rotational invariance in the GP formulation, we consider the representation of the matrix exponential

$$\mathbf{S} = \exp(\mathbf{B}) = \mathbf{Q}\exp(\boldsymbol{\Lambda})\mathbf{Q}^T, \tag{26}$$

from a limited number of training samples. As in Section 4.2, $\mathbf{B} = \mathbf{Q}\boldsymbol{\Lambda}\mathbf{Q}^T$ is a symmetric diagonalizable matrix with eigenvector matrix $\mathbf{Q}$ and diagonal eigenvalue matrix $\boldsymbol{\Lambda}$. Clearly the matrix exponential maintains rotational invariance under application of a rotation $\mathbf{R}$, and the series representation of $\exp(\mathbf{B})$ demonstrates the coaxiality of $\exp(\mathbf{B})$ and $\mathbf{B}$. Given Eq. (23), the expansion coefficients may be determined from

$$\begin{bmatrix} \exp\lambda_1 \\ \exp\lambda_2 \\ \exp\lambda_3 \end{bmatrix} = \begin{bmatrix} 1 & \lambda_1 & \lambda_1^2 \\ 1 & \lambda_2 & \lambda_2^2 \\ 1 & \lambda_3 & \lambda_3^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}. \tag{27}$$

To create the dataset, we draw random uniformly distributed 3×3 matrices with entries between $[0, 1]$ and compute their symmetric part. A single GP is formed over the six independent tensor components (the upper triangular part) of $\mathbf{B}$ to predict the six independent output tensor components. This formulation is compared against the TBGP formulation, where the three invariants are used to predict the three expansion coefficients with a single GP. The *scikit-learn* library (Pedregosa et al., 2011) was used to train the GPs in both cases. The root-mean-square error is evaluated in each case at 10,000 testing points for validation. The input training points are then rotated randomly, and the GP prediction is evaluated at the inputs.

Figure 1 shows the results of the testing error as a function of increasing training points for the GP and TBGP formulations. Since the rotationally invariant formulation does not take the orientation of the eigenvectors into account, it is expected that the prediction error on the modified inputs in this case would be small, whereas the prediction error for the normal GP could be quite large. The TBGP error is indeed 1–2 orders of magnitude lower than the GP error on the testing sets, and the error on the randomly rotated training set is also over 5 orders of magnitude lower, demonstrating that the TBGP formulation learns the underlying function much more quickly than a standard GP. It also appears to take an order of magnitude order more data before the GP error catches up to the error that the TBGP had on the smallest dataset. The error on the rotated training set appears to increase, which we attribute to a combination of increasing condition number of the covariance matrix and accumulated interpolation error in the GP through the training points from the use of regularization to maintain invertibility. Even with these effects, the TBGP has very little error throughout the tests and is uniformly the better choice.

5. STRAIN ENERGY POTENTIAL GAUSSIAN PROCESS

The tensor basis GP formulation is quite powerful and could be made general to many different types of processes. For hyperelastic materials, where the stress is the derivative of a potential, it is possible to take this approach one step further with an alternate formulation. As in Eq. (3), we

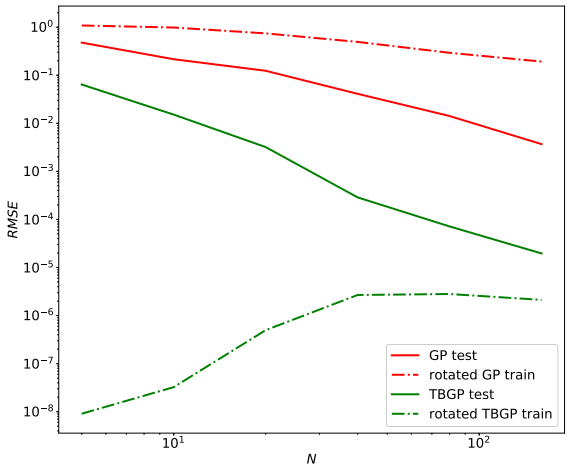


FIG. 1: The root-mean-square error (RMSE) in predicting the matrix exponential for the GP and TBGP formulations as a function of training set size. The results also show the RMSE of the regressors evaluated at random rotations of the training set pairs.

define the strain-energy density function Φ such that the stress tensor may be computed as

$$\boldsymbol{\sigma} = \frac{2}{I_3^{1/2}}\mathbf{B}\frac{\partial \Phi}{\partial \mathbf{B}}, \tag{28}$$

for some appropriate $\Phi(\mathbf{B})$. The strain-energy density is an invariant scalar quantity and cannot depend on the rotation of the frame; thus for an isotropic hyperelastic material it is preferable to express the strain-energy density as a function of the invariants of $\mathbf{B}$. Thus the expression for the stress tensor may be expanded as

$$\boldsymbol{\sigma} = \frac{2}{I_3^{1/2}}\mathbf{B}\left(\frac{\partial \Phi}{\partial I_1}\frac{\partial I_1}{\partial \mathbf{B}} + \frac{\partial \Phi}{\partial I_2}\frac{\partial I_2}{\partial \mathbf{B}} + \frac{\partial \Phi}{\partial I_3}\frac{\partial I_3}{\partial \mathbf{B}}\right), \tag{29}$$

where we have applied the chain rule for the strain-energy density partial derivatives. The derivatives of the invariants with respect to the original tensor may be computed directly using matrix calculus identities (Bonet and Wood, 1997). Specifically, we use

$$\begin{aligned}
\frac{\partial I_1}{\partial \mathbf{B}} &= \mathbf{I}, \\
\frac{\partial I_2}{\partial \mathbf{B}} &= I_1\mathbf{I} - \mathbf{B}, \\
\frac{\partial I_3}{\partial \mathbf{B}} &= I_3\mathbf{B}^{-1},
\end{aligned} \tag{30}$$

and the expression for $\boldsymbol{\sigma}$ simplifies to

$$\boldsymbol{\sigma} = \frac{2}{I_3^{1/2}}\left[I_3\frac{\partial \Phi}{\partial I_3}\mathbf{I} + \left(\frac{\partial \Phi}{\partial I_1} + I_1\frac{\partial \Phi}{\partial I_2}\right)\mathbf{B} - \frac{\partial \Phi}{\partial I_2}\mathbf{B}^2\right]. \tag{31}$$

This expression explicitly takes the form of a tensor-basis expansion of the type in Eq. (17), where we have

$$\begin{aligned}
c_0 &= 2I_3^{1/2}\frac{\partial \Phi}{\partial I_3}, \\
c_1 &= \frac{2}{I_3^{1/2}}\left(\frac{\partial \Phi}{\partial I_1} + I_1\frac{\partial \Phi}{\partial I_2}\right), \\
c_2 &= -\frac{2}{I_3^{1/2}}\frac{\partial \Phi}{\partial I_2}.
\end{aligned} \tag{32}$$

For a given set of coefficients and invariants, the corresponding partial derivatives can be evaluated using the following relations:

$$\begin{aligned}
\frac{\partial \Phi}{\partial I_1} &= \frac{I_3^{1/2}}{2}(c_1 + c_2 I_1), \\
\frac{\partial \Phi}{\partial I_2} &= -\frac{c_2 I_3^{1/2}}{2}, \\
\frac{\partial \Phi}{\partial I_3} &= \frac{c_0}{2I_3^{1/2}}.
\end{aligned} \tag{33}$$

Thus given a set of observed pairs of $(\boldsymbol{\sigma}, \mathbf{B})$ and Eqs. (23) and (33), we can infer the corresponding values of the gradient of the strain-energy density function. Furthermore, we "ground" the function (i.e., remove the indeterminacy of calibration of Φ from derivative data) by choosing a zero-point energy for a set of invariants where the material has not been deformed. Hence, we augment the gradient information with the datum corresponding to $\boldsymbol{\sigma}(\mathbf{B} = \mathbf{I}) = 0$:

$$\Phi(I_1 = 3, I_2 = 3, I_3 = 1) = 0. \tag{34}$$

The GP regression technique can be extended to take advantage of derivative information since the derivative of a GP is also a GP. Specifically, the covariance between the derivatives of the stress-energy density between two different points in invariant space, $\mathbb{I} = (I_1, I_2, I_3)$ and $\mathbb{I}' = (I_1', I_2', I_3')$, can be evaluated as

$$\mathrm{Cov}\left[\frac{\partial \Phi(\mathbb{I})}{\partial I_i}, \Phi(\mathbb{I}')\right] = \frac{\partial K(\mathbb{I}, \mathbb{I}')}{\partial I_i}, \tag{35}$$

$$\mathrm{Cov}\left[\frac{\partial \Phi(\mathbb{I})}{\partial I_i}, \frac{\partial \Phi(\mathbb{I}')}{\partial I_j'}\right] = \frac{\partial^2 K(\mathbb{I}, \mathbb{I}')}{\partial I_i \partial I_j'}. \tag{36}$$

Using these relations, a GP may be formed simultaneously over Φ at the grounding point and its derivatives over the dataset by using the block covariance matrix

$$\mathsf{K}_\Phi(\mathbb{I}, \mathbb{I}') = \begin{bmatrix} \dfrac{\partial^2 K}{\partial I_1 \partial I_1'} & \dfrac{\partial^2 K}{\partial I_1 \partial I_2'} & \dfrac{\partial^2 K}{\partial I_1 \partial I_3'} \\ \dfrac{\partial^2 K}{\partial I_2 \partial I_1'} & \dfrac{\partial^2 K}{\partial I_2 \partial I_2'} & \dfrac{\partial^2 K}{\partial I_2 \partial I_3'} \\ \dfrac{\partial^2 K}{\partial I_3 \partial I_1'} & \dfrac{\partial^2 K}{\partial I_3 \partial I_2'} & \dfrac{\partial^2 K}{\partial I_3 \partial I_3'} \end{bmatrix}, \tag{37}$$

where each entry is a matrix corresponding to the covariance between the individual derivatives of Φ evaluated between each of the training data points. The grounding point $\mathbb{I}_g = (3, 3, 1)$ is included by augmenting this matrix with an additional row and column:

$$\mathsf{K}_g(\mathbb{I}, \mathbb{I}') = \begin{bmatrix} K(\mathbb{I}_g, \mathbb{I}_g) & \dfrac{\partial K(\mathbb{I}_g, \mathbb{I}')}{\partial I_1'} & \dfrac{\partial K(\mathbb{I}_g, \mathbb{I}')}{\partial I_2'} & \dfrac{\partial K(\mathbb{I}_g, \mathbb{I}')}{\partial I_3'} \\ \dfrac{\partial K(\mathbb{I}, \mathbb{I}_g)}{\partial I_1} & & & \\ \dfrac{\partial K(\mathbb{I}, \mathbb{I}_g)}{\partial I_2} & & \mathsf{K}_\Phi(\mathbb{I}, \mathbb{I}') & \\ \dfrac{\partial K(\mathbb{I}, \mathbb{I}_g)}{\partial I_3} & & & \end{bmatrix}. \tag{38}$$

In a slight abuse of notation, in the matrix $\mathsf{K}_\Phi(\mathbb{I}, \mathbb{I}')$ in Eq. (37) and the matrix $\mathsf{K}_g(\mathbb{I}, \mathbb{I}')$ in Eq. (38), the arguments $\mathbb{I}$ and $\mathbb{I}'$ should be interpreted as matrices of the invariants at the training points (as opposed to individual points).

The GP mean of the potential, Φ, and its gradient with respect to the invariants, $\nabla_{\mathbb{I}}\Phi = \{\frac{\partial\Phi}{\partial I_1}, \frac{\partial\Phi}{\partial I_2}, \frac{\partial\Phi}{\partial I_3}\}$, at a new point $\mathbb{I}^* = (I_1^*, I_2^*, I_3^*)$ may then be predicted with

$$\mathbb{E}[(\Phi(\mathbb{I}^*), \nabla_{\mathbb{I}}\Phi(\mathbb{I}^*)] = \mathsf{K}_g(\mathbb{I}^*, \mathbb{I})\,\mathsf{w}, \tag{39}$$

where we have defined the weight vector

$$\mathsf{w} = \mathsf{K}_g^{-1}(\mathbb{I}, \mathbb{I}') \begin{bmatrix} 0 & \dfrac{\partial\Phi}{\partial I_1} & \dfrac{\partial\Phi}{\partial I_2} & \dfrac{\partial\Phi}{\partial I_3} \end{bmatrix}^T, \tag{40}$$

where the right-hand side partial derivatives are at the observed data points, and $\mathsf{K}_g(\mathbb{I}^*, \mathbb{I})$ is the covariance between the test points and the training points augmented with the ground point. This expression is analogous to Eq. (10), although here we have omitted the noise term. The gradient of the potential Φ, and hence the stress $\boldsymbol{\sigma}$, may be evaluated using the same weight vector.

Although this formulation sets a coregionalization model for multioutput GPR, the inclusion of derivative information in the covariance matrix has a tendency to exacerbate ill-conditioning and round-off errors in the inference process (Wu et al., 2017). Thus it is expected that prediction accuracy may deteriorate in certain circumstances.

6. RESULTS: MOONEY–RIVLIN MATERIAL

In this section we consider the application of GPR to predicting data drawn from the stress response of the deformation of a hyperelastic material. The underlying truth model will be assumed to be a compressible Mooney–Rivlin material with strain-energy density function

$$\Phi = k_1\big(I_3^{-1/2}I_1 - 3\big) + k_2\big(I_3^{-2/3}I_2 - 3\big) + k_3\big(I_3^{1/2} - 1\big)^2, \tag{41}$$

where we take $k_1 = 0.162$ MPa, $k_2 = 0.0059$ MPa, and $k_3 = 10$ MPa (Marckmann and Verron, 2006) and we make k_3 large ($k_3 \gg k_1, k_2$) to effect a nearly incompressible material response.

We generate realizations of arbitrary mixed compression/tension/shear states using the following procedure. Let $\mathbf{V}$ be a diagonal matrix of randomly sampled positive values between $0 < l \leq 1 < u$, and let $\mathbf{R}$ be a random rotation matrix sampled uniformly on SO(3). We then employ the polar decomposition of the deformation gradient tensor

$$\mathbf{F} = \mathbf{R}\mathbf{V}, \tag{42}$$

with corresponding Finger tensor

$$\mathbf{B} = \mathbf{R}\mathbf{V}^2\mathbf{R}^T, \tag{43}$$

thus guaranteeing that the determinant of $\mathbf{F}$ is positive and that $\mathbf{B}$ corresponds to a valid diagonalizable tensor with some superposition of tension/compression and shear in arbitrary directions. The corresponding eigenvalues of $\mathbf{B}$ are randomly distributed in the interval $[l^2, u^2]$.

The results shown in Figs. 2, 3, and 4 are for two datasets, one sampling $[l^2, u^2] = [1.0, 1.5]$ and another covering $[0.9, 2.0]$. This first choice includes strain values corresponding only to mild extension along the principal axes, while the second choice includes a more extreme range from mild compression to much more extension. The GP, TBGP, and potential-TBGP formulations were trained on 100 different random samples of datasets of varying size, and each trial's hyperparameters were selected with multistart L-BFGS optimization (Byrd et al., 1995) of the marginal likelihood with 20 random initializations.

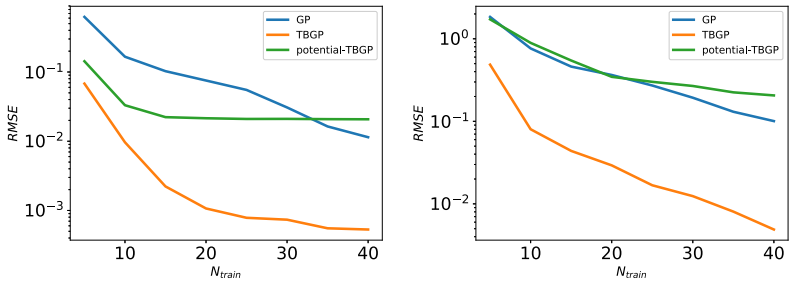


FIG. 2: The root-mean-square error of the stress tensor for the GP, TBGP, and potential-TBGP formulations as a function of training set size for $[l^2, u^2] = [1.0, 1.5]$ (upper panel) and $[0.9, 2.0]$ (lower panel)

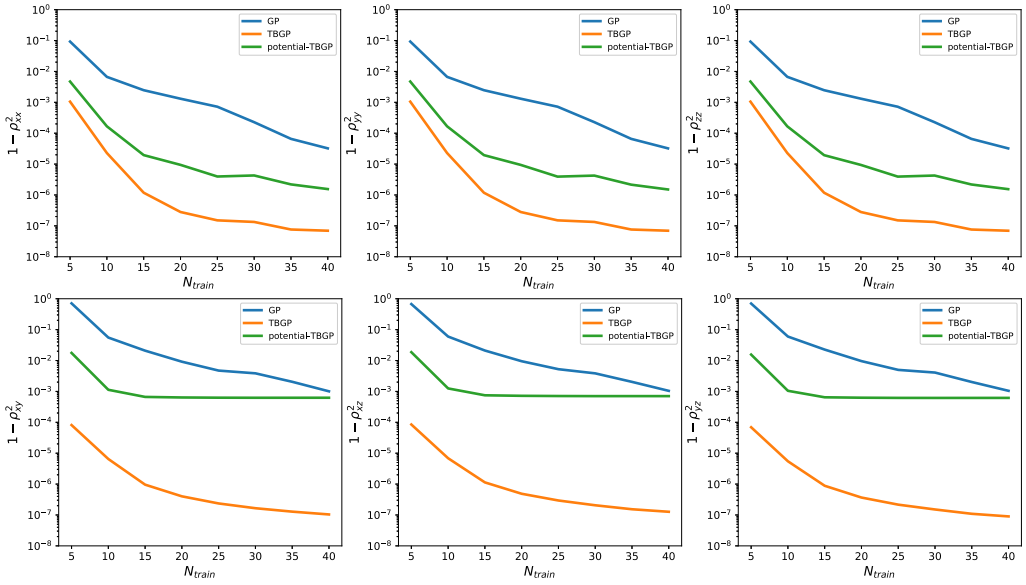


FIG. 3: The component-wise fraction of variance unexplained $(1 - \rho^2)$ for the stress tensor predictions as a function of training set size for $[l^2, u^2] = [1.0, 1.5]$. Top row are the tension components, and the bottom are the shear components.

The root-mean-square error is shown in Fig. 2 and fraction of variance unexplained $1 - \rho^2$ for correlation coefficient ρ for the stress tensor components are shown in Figs. 3 and 4 as a function of the training set size for both datasets. It is clear that the TBGP has a substantially lower error than the GP with approximately the same rate of convergence, with 1–2 orders of magnitude lower error and 5–6 orders of magnitude lower unexplained variance. As in the matrix exponential example, the TBGP formulation is uniformly the best choice at all tested numbers of datapoints. However, the potential-based TBGP has about the same error as the regular GP, and both show slightly degraded performance on predicting the shear components of the stress tensor compared to the tension components. In addition, for larger sets of data the accuracy of potential-TBGP stops improving and, in the larger deformation case, the error diverges. We attribute this trend to attempting to learn the full function behavior from gradient information alone, as well as the much larger and more ill-conditioned covariance matrix formed in the

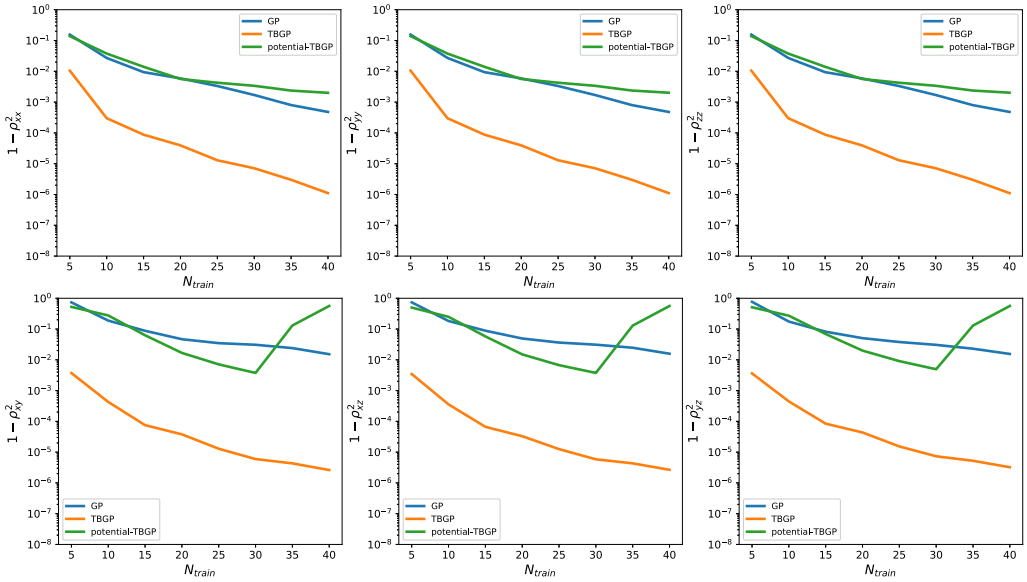


FIG. 4: The component-wise fraction of variance unexplained ($1-\rho^2$) for the stress tensor predictions as a function of training set size for $[l^2, u^2] = [0.9, 2.0]$. Top row are the tension components, and the bottom are the shear components.

inference process. Individual trials of the training process become more likely to yield higher-error models, increasing the average error. It is likely that using advanced low-rank factorizations of the covariance matrix or better regularization would reduce the magnitude of this diverging error.

Although the potential-TBGP performance for predicting the stress components is no better than the GP, it is capable of predicting the strain-energy density function with fairly high accuracy, which is not a task that either of the other two formulations are capable of doing directly. Figure 5 shows the root-mean-square error and correlation coefficient as a function of training set size. The prediction accuracy does show substantial improvement with increasing training data for the lower-stretch case $[l^2, u^2] = [1.0, 1.5]$, but the ill-conditioning issues prevalent in the stress predictions for the higher-stretch case $[l^2, u^2] = [0.9, 2.0]$ pervade the potential prediction as well. Figure 6 shows that the source of disagreement between the potential-TBGP and the Mooney–Rivlin potential originates from values at higher energy, where there is less training data available and a more complex trend to predict. One possible solution to improve the performance of the potential-TBGP is to use a greedy point selection approach that would use points that span a wider range of invariant space and energy density to train the GP.

7. CONCLUSION

This paper has developed and demonstrated an approach to embedding rotational invariance in the GPR framework for constitutive modeling of isotropic hyperelasticity. Embedding this physics knowledge led to a dramatic improvement in the accuracy and learning curves for the TBGP formulation compared to the traditional component-based approach. Also the potential-TBGP formulation demonstrated recovery of the potential function from stress-strain data with

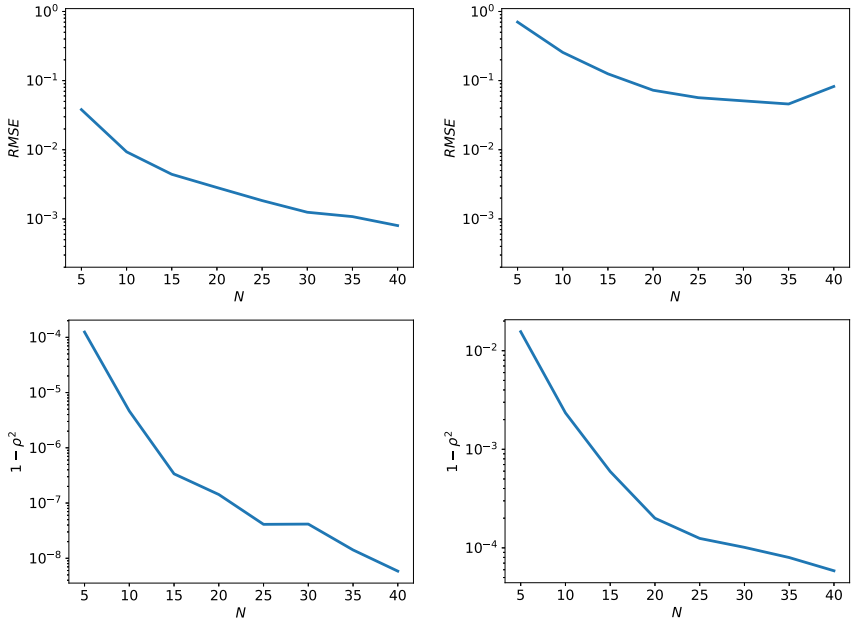


FIG. 5: The root-mean-square error (upper) and unexplained variance (lower) of the strain energy density function Φ as a function of training set size from the potential-TBGP for $[l^2, u^2] = [1.0, 1.5]$ (left) and $[l^2, u^2] = [0.9, 2.0]$ (right)

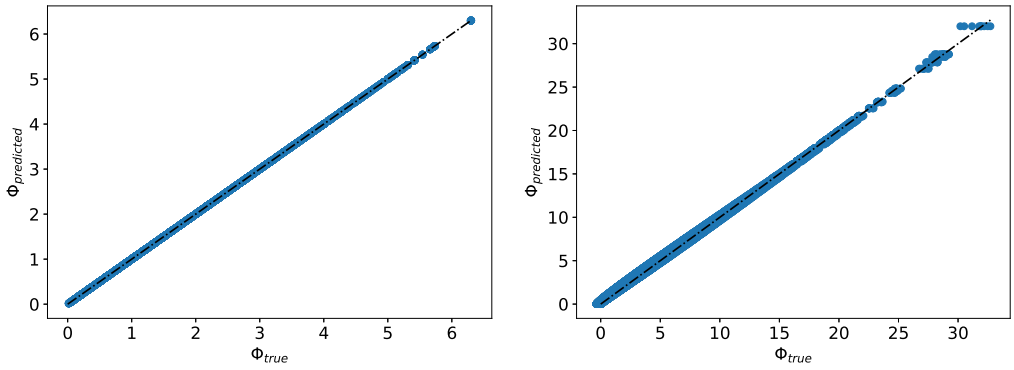


FIG. 6: Scatter-plot of the predicted and actual strain energy density values at the highest training set size for the cases $[l^2, u^2] = [1.0, 1.5]$ (upper panel) and $[0.9, 2.0]$ (lower panel)

comparable accuracy to the plain GP formulation for stress prediction. While the examples considered here are relatively simple, the application of the methodology to more complex hyperelastic materials and functions of kinematic variables is straightforward.

One important consideration for future work is the representation of anisotropic material response and functions of multiple tensors. In these cases, the tensor basis and corresponding invariants are more complex, but the underlying process remains the same. For example, the second Piola–Kirchoff tensor $\mathbf{S}$ may be expressed as a function of the Cauchy–Green deformation tensor $\mathbf{C} = \mathbf{F}^T\mathbf{F}$ and a structure tensor $\mathbf{G}$ that characterizes the anisotropy in the response:

$$\mathbf{S} = \mathbf{f}(\mathbf{C}, \mathbf{G}), \tag{44}$$

[see Zheng (1994) for more details]. For the case of transverse isotropy where the material response along a direction $\mathbf{g}$ is different than in the plane perpendicular to the unit vector $\mathbf{g}$, $\mathbf{G} = \mathbf{g} \otimes \mathbf{g}$ can be employed as the structure tensor. The corresponding expansion for $\mathbf{S}$ is

$$\mathbf{S} = \sum_{i=1}^{6} c_i \mathbf{A}_i, \tag{45}$$

in terms of the tensors $\mathbf{A}_i \in \{\mathbf{I}, \mathbf{C}, \mathbf{C}^2, \mathbf{G}, \mathbf{CG} + \mathbf{GC}, \mathbf{C}^2\mathbf{G} + \mathbf{GC}^2\}$ and coefficients c_i, which are functions of the extended invariant set $\{\mathrm{tr}\,\mathbf{C}, \mathrm{tr}\,\mathbf{C}^2, \mathrm{tr}\,\mathbf{C}^3, \mathrm{tr}\,\mathbf{CG}, \mathrm{tr}\,\mathbf{C}^2\mathbf{G}\}$ [refer to Boehler (1987) and use $\mathbf{G}^2 = \mathbf{G}$]. Since $\mathbf{S}$ is a symmetric tensor and the tensor basis $\{\mathbf{A}_i\}$ is a linearly independent set, the expansion gives six equations for the six unknowns c_i, which may be solved readily. (If representation theory suggests more tensor basis elements than unknowns, the basis can be reduced to an independent set that spans the output, at which point the suggested procedure applies.) In this way, the corresponding coefficients as a function of the invariants may be inferred and a TBGP may be trained to make predictions for an anisotropic material response.

There is also room to improve the predictions of the potential-based TBGP. The squared-exponential kernel was selected for its simplicity and smoothness, but it is possible that a more complex or nonstationary kernel would be able to capture the behavior of the potential function in invariant-space more accurately and overcome the ill-conditioning issues seen in this formulation. Other work suggests that approaches involving low-rank or spectral representations of the covariance matrix with derivative information would likely improve the accuracy of the inference process as well.

ACKNOWLEDGMENTS

This work was supported by the LDRD program at Sandia National Laboratories, and its support is gratefully acknowledged. Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525. The views expressed in the article do not necessarily represent the views of the U.S. Department of Energy or the U.S. government. This paper has been deemed UUR with SAND number SAND2020-5235J.

REFERENCES

Bachoc, F., Lagnoux, A., and López-Lopera, A., Maximum Likelihood Estimation for Gaussian Processes under Inequality Constraints, *Elect. J. Stat.*, vol. **13**, no. 2, pp. 2921–2969, 2019.

Boehler, J., Representations for Isotropic and Anisotropic Non-Polynomial Tensor Functions, in *Applications of Tensor Functions in Solid Mechanics*, pp. 31–53, Berlin: Springer, 1987.

Bonet, J. and Wood, R., *Nonlinear Continuum Mechanics for Finite Element Analysis*, Cambridge, UK: Cambridge University Press, 1997.

Brunton, S.L., Proctor, J.L., and Kutz, J.N., Discovering Governing Equations from Data by Sparse Identification of Nonlinear Dynamical Systems, *Proc. Nat. Acad. Sci.*, vol. **113**, no. 15, pp. 3932–3937, 2016.

Byrd, R., Lu, P., and Nocedal, J., A Limited Memory Algorithm for Bound Constrained Optimization, *SIAM J. Sci. Stat. Comput.*, vol. **16**, pp. 1190–1208, 1995.

Da Veiga, S. and Marrel, A., Gaussian Process Modeling with Inequality Constraints, *Annales de la Faculté des Sciences de Toulouse*, vol. **21**, pp. 529–555, 2012.

Dean, J., Corrado, G., Monga, R., Chen, K., Devin, M., Mao, M., Ranzato, M., Senior, A., Tucker, P., and Yang, K., Large Scale Distributed Deep Networks, *Adv. Neural Inf. Process. Syst.*, pp. 1223–1231, 2012.

Doyle, T. and Ericksen, J., Nonlinear Elasticity, *Adv. Appl. Mech.*, vol. **4**, pp. 53–115, 1956.

Frankel, A., Jones, R., Alleman, C., and Templeton, J., Predicting the Mechanical Response of Oligocrystals with Deep Learning, *Comput. Mater. Sci.*, vol. **169**, p. 109099, 2019a.

Frankel, A., Tachida, K., and Jones, R., Prediction of the Evolution of the Stress Field of Polycrystals Undergoing Elastic-Plastic Deformation with a Hybrid Neural Network Nodel, 2019b. arXiv: 1910.03172

Gurtin, M., *An Introduction to Continuum Mechanics*, vol. **158**, Cambridge, MA: Academic Press, 1982.

Hastie, T., Tibshirani, R., and Friedman, J., *The Elements of Statistical Learning: Data Mining, Inference and Prediction*, 2nd Edition, Berlin: Springer, 2016.

Hill, R., On Constitutive Inequalities for Simple Materials—I, *J. Mech. Phys. Solids*, vol. **16**, no. 4, pp. 229–242, 1968.

Jensen, B., Nielsen, J., and Larsen, J., Bounded Gaussian Process Regression, in *2013 IEEE Int. Workshop on Machine Learning for Signal Processing (MLSP)*, Southampton, UK, 2013.

Jones, R., Templeton, J., Sanders, C., and Ostien, J., Machine Learning Models of Plastic Flow based on Representation Theory, *Comput. Model. Eng. Sci.*, pp. 309–342, 2018.

Karpatne, A., Atluri, G., Faghmous, J., Steinbach, M., Banerjee, A., Ganguly, A., Shekhar, S., Samatova, N., and Kumar, V., Theory-Guided Data Science: A New Paradigm for Scientific Discovery from Data, *IEEE Transact. Knowledge Data Eng.*, vol. **29**, no. 10, pp. 2318–2331, 2017.

Lee, K. and Carlberg, K., Model Reduction of Dynamical Systems on Nonlinear Manifolds Using Deep Convolutional Autoencoders, *J. Comput. Phys.*, vol. **404**, p. 108973, 2020.

Ling, J., Jones, R., and Templeton, J., Machine Learning Strategies for Systems with Invariance Properties, *J. Comput. Phys.*, vol. **318**, pp. 22–35, 2016.

López-Lopera, A., Bachoc, F., Durrande, N., and Roustant, O., Finite-Dimensional Gaussian Approximation with Linear Inequality Constraints, *SIAM/ASA J. Uncertain. Quant.*, vol. **6**, no. 3, pp. 1224–1255, 2018.

Lusch, B., Kutz, J.N., and Brunton, S.L., Deep Learning for Universal Linear Embeddings of Nonlinear Dynamics, *Nat. Commun.*, vol. **9**, no. 1, p. 4950, 2018.

Magiera, J., Ray, D., Hesthaven, J., and Rohde, C., Constraint-Aware Neural Networks for Riemann Problems, *J. Comput. Phys.*, vol. **409**, p. 109345, 2020.

Malvern, L., *Introduction to the Mechanics of a Continuous Medium*, Engle Cliffs, NJ: Prentice Hall Inc., 1969.

Marckmann, G. and Verron, E., Comparison of Hyperelastic Models for Rubber-Like Materials, *Rubber Chem. Technol.*, vol. **79**, no. 5, pp. 835–858, 2006.

Ogden, R., *Non-Linear Elastic Deformations*, North Chelmsford, MA: Courier Corporation, 1997.

Pan, S. and Duraisamy, K., Data-Driven Discovery of Closure Models, *SIAM J. Appl. Dyn. Syst.*, vol. **17**, no. 4, pp. 2381–2413, 2018.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E., Scikit-Learn: Machine Learning in Python, *J. Mach. Learn. Res.*, vol. **12**, pp. 2825–2830, 2011.

Raissi, M. and Karniadakis, G., Hidden Physics Models: Machine Learning of Nonlinear Partial Differential Equations, *J. Comput. Phys.*, vol. **357**, pp. 125–141, 2018.

Raissi, M., Perdikaris, P., and Karniadakis, G., Machine Learning of Linear Differential Equations Using Gaussian Processes, *J. Comput. Phys.*, vol. **348**, pp. 683–693, 2017.

Rasmussen, C. and Williams, C., *Gaussian Processes for Machine Learning*, Cambridge, MA: MIT Press, 2006.

Riihimäki, J. and Vehtari, A., Gaussian Processes with Monotonicity Information, in *Proc. of the Thirteenth Int. Conf. on Artificial Intelligence and Statistics*, Sardinia, Italy, 2010.

Solak, E., Murray-Smith, R., Leithead, W., Leith, D., and Rasmussen, C., Derivative Observations in Gaussian Process Models of Dynamic Systems, in *Advances in Neural Information Processing Systems*, pp. 1057–1064, 2003.

Wu, A., Aoi, M.C., and Pillow, J.W., Exploiting Gradients and Hessians in Bayesian Optimization and Bayesian Quadrature, 2017. arXiv: 1704.00060

Yang, X., Tartakovsky, G., and Tartakovsky, A., Physics-Informed Kriging: A Physics-Informed Gaussian Process Regression Method for Data-Model Convergence, 2018. arXiv: 1809.03461

Zheng, Q., Theory of Representations for Tensor Functions—A Unified Invariant Approach to Constitutive Equations, *Appl. Mech. Rev.*, vol. **47**, no. 11, pp. 545–587, 1994.

LIMITATIONS OF PHYSICS INFORMED MACHINE LEARNING FOR NONLINEAR TWO-PHASE TRANSPORT IN POROUS MEDIA

***Olga Fuks**[*] & **Hamdi A. Tchelepi**[*]*

Department of Energy Resources Engineering, Stanford University, 367 Panama Street, Stanford, California 94305, USA

*Address all correspondence to: Hamdi A. Tchelepi or Olga Fuks, Department of Energy Resources Engineering, Stanford University, 367 Panama Street, Stanford, California 94305, USA, E-mail: tchelepi@stanford.edu; E-mail: ofuks@stanford.edu

Original Manuscript Submitted: 2/11/2020; Final Draft Received: 6/10/2020

Deep learning techniques have recently been applied to a wide range of computational physics problems. In this paper, we focus on developing a physics-based approach that enables the neural network to learn the solution of a dynamic fluid-flow problem governed by a nonlinear partial differential equation (PDE). The main idea of physics informed machine learning (PIML) approaches is to encode the underlying physical law (i.e., the PDE) into the neural network as prior information. We investigate the applicability of the PIML approach to the forward problem of immiscible two-phase fluid transport in porous media, which is governed by a nonlinear first-order hyperbolic PDE subject to initial and boundary data. We employ the PIML strategy to solve this forward problem without any additional labeled data in the interior of the domain. Particularly, we are interested in nonconvex flux functions in the PDE, where the solution involves shocks and mixed waves (shocks and rarefactions). We have found that such a PIML approach fails to provide reasonable approximations to the solution in the presence of shocks in the saturation field. We investigated several architectures and experimented with a large number of neural-network parameters, and the overall finding is that PIML strategies that employ the nonlinear hyperbolic conservation equation in the loss function are inadequate. However, we have found that employing a parabolic form of the conservation equation, whereby a small amount of diffusion is added, the neural network is consistently able to learn accurate approximation of the solutions containing shocks and mixed waves.

KEY WORDS: *two-phase transport, physics informed machine learning, partial differential equations*

1. INTRODUCTION

Machine learning (ML) techniques, specifically deep learning (LeCun et al., 2015), are at the center of attention across the computational science and engineering communities. The spectrum of deep learning architectures and techniques has already achieved notable results across applications and disciplines, including computer vision and image recognition (He et al., 2016; Karpathy et al., 2014; Krizhevsky et al., 2012), speech recognition and machine translation (Hinton et al., 2012; Sutskever et al., 2014), robotics (Lillicrap et al., 2015; Mnih et al., 2016), and

2689–3967/20/$35.00

medicine (Gulshan et al., 2016; Liu et al., 2017). There is no doubt that the range of applications will grow and the impact of ML methods will continue to spread.

Deep learning allows neural networks composed of multiple processing layers to learn representations of raw input data with multiple levels of abstraction. These networks are known to be particularly effective at supervised learning tasks, whereby the successful application of these models usually requires the availability of large amounts of labeled data. However, in many engineering applications, data acquisition is often prohibitively expensive, and the amount of labeled data is usually quite sparse. Specifically, most computational geoscience problems related to modeling subsurface flow dynamics suffer from sparse site-specific data. Consequently, in this "sparse data" regime, it is crucial to employ domain knowledge to reduce the need for labeled training data, or even aim to train ML models without any labeled data relying only on constraints (Stewart and Ermon, 2017). These constraints are used to encode the specific structure and properties of the output that are known to hold because of domain knowledge, e.g., known physics laws such as conservation of momentum, mass, and energy.

Physics informed machine learning approaches have been explored recently in a variety of computational physics problems, whereby the focus is on enabling the neural network to learn the solutions of deterministic partial differential equations (PDEs). Early works in this area date back to the 1990s (Lagaris et al., 1998; Lee and Kang, 1990; Meade Jr. and Fernandez, 1994; Psichogios and Ungar, 1992). However, in the context of modern neural network architectures, the interest in this topic has been revived (Raissi et al., 2017, 2019; Zhu et al., 2019). These so-called physics informed machine learning (PIML) approaches are designed to obtain data-driven solutions of general nonlinear PDEs, and they may be a promising alternative to traditional numerical methods for solving PDEs, such as finite-difference and finite-volume methods. The core idea of PIML is that the developed neural network encodes the underlying physical law as prior information, and then uses this information during the training process. The approach takes advantage of the neural network capability to approximate any continuous function (Cybenko, 1989; Hornik et al., 1989). Raissi et al. (2017) demonstrated the PIML capabilities for a collection of diverse problems in computational science (Burgers' equation, Navier-Stokes, etc.). They suggested that if the considered PDE is well-posed and its solution is unique, then the PIML method is capable of achieving good predictive accuracy given a sufficiently expressive neural network architecture and a sufficient number of collocation points. In the current work, we show that the neural network approach struggles and even fails for modeling the nonlinear hyperbolic PDE that governs two-phase transport in porous media. Our experience indicates that this shortcoming of PIML for hyperbolic PDEs is not related to the specific architecture, or to the choice of the hyperparameters (e.g., number of collocation points, etc.).

One important class of PDEs is that of conservation laws that describe the conservation of mass, momentum, and energy. In particular, these conservation equations describe displacement processes that are essential for modeling flow and transport in subsurface porous formations, such as water-oil or gas-oil displacements (Aziz and Settari, 1979; Orr, 2007). Numerical reservoir simulation based on solving mass conservation equations with constitutive relations for the nonlinear coefficients is used to make predictions. A major challenge in practice is that the available information/measurements (i.e., labeled data) about the specific geological formation of interest is often quite sparse. Thus, it is critical to take advantage of any prior information in order to improve the predictive reliability of the computational models. The physics of two-phase fluid transport, e.g., water-oil displacements, is described by a nonlinear hyperbolic PDE [or a system of PDEs (Orr, 2007)]. These nonlinear transport problems are known to be quite challenging for standard numerical methods (Aziz and Settari, 1979), and this is largely due

to the presence of steep saturation fronts and mixed waves (shocks and spreading waves) in the solution. Specifically, we are interested in solving Riemann problems—initial value problems, when the initial data consist of two constant states separated by a jump discontinuity at $x = 0$.

There are significant efforts aimed at figuring out the potential of machine learning in the modeling of flow processes in large-scale subsurface formations. Thus, it is extremely important to understand the limitations of PIML schemes for making computational predictions of reservoir displacement processes. Here, we investigate the application of the physics informed machine learning approach to the "pure" forward problem of nonlinear two-phase transport in porous media. We evaluate the performance of the PIML framework for this problem with different flux (fractional flow) functions. The objective is to assess how well this PIML approach performs for nonlinear flow problems with discontinuous solutions (i.e., shocks).

The paper proceeds as follows. In Section 2, we describe the two-phase transport model and the governing hyperbolic PDE that we aim to solve with a machine learning approach. In Section 3 we provide a brief overview of the physics informed machine learning framework that we use to solve the deterministic PDE. The results for the transport problem with different flux functions are presented in Section 4. Then, to understand the observed behavior of the method we provide a more detailed analysis of the trained neural networks in Section 5. Lastly, in Section 6, we summarize our findings and provide a brief discussion of the results.

2. TWO-PHASE TRANSPORT MODEL

We consider the standard Buckley-Leverett model with two incompressible immiscible fluids, e.g., oil and water. A nonwetting phase, e.g., oil (o), is displaced by a wetting phase, e.g., water (w), in a porous medium with permeability $k(\mathbf{x})$ and porosity $\phi(\mathbf{x})$. Gravity and capillary effects are neglected. Under these assumptions, the pressure p and fluid saturations S_α ($\alpha = o, w$) are governed by a coupled system of mass balance equations complemented by Darcy's equations for each phase. After some manipulation [see, e.g., Aziz and Settari (1979)], the system can be transformed into the incompressibility condition for the total flux, $\mathbf{u}_{tot}$:

$$\nabla \cdot \mathbf{u}_{tot} = q_t, \tag{1}$$

where q_t is a total source (sink) term, and the conservation equation for one of the phases, e.g., water:

$$\phi(\mathbf{x})\frac{\partial S_w}{\partial t} + \nabla \cdot (f_w(S_w) \cdot \mathbf{u}_{tot}) = q_w. \tag{2}$$

Here $\mathbf{u}_{tot} = \mathbf{u}_w + \mathbf{u}_o$ is the total flux and $\mathbf{u}_\alpha$ represents the Darcy's flux for a phase ($\alpha = \mathrm{o}, \mathrm{w}$); the function f_w is called the fractional flow of water or simply, flux function, and is defined as follows:

$$f_w = \frac{\lambda_w}{\lambda_w + \lambda_o}, \tag{3}$$

where $\lambda_\alpha = (k\, k_{r\alpha})/\mu_\alpha$ stands for the phase mobility, μ_α is the viscosity of the phase, $k_{r\alpha}(S_\alpha)$ is the relative phase permeability, and q_w is a source (sink) term for water. The source or sink terms represent the effect of wells. Equation (2) is supplemented with uniform initial and boundary conditions:

$$\begin{aligned} S_w(\mathbf{x}, t) &= s_{wi}, \quad \forall \mathbf{x} \quad \text{and} \quad t = 0, \\ S_w(\mathbf{x}, t) &= s_b, \quad \mathbf{x} \in \Gamma_{inj} \quad \text{and} \quad t > 0, \end{aligned} \tag{4}$$

where s_{wi} is the initial water saturation in the reservoir, and s_b is the saturation at the injection well or boundary, Γ_{inj}.

In one-dimensional space, Eq. (2) becomes

$$\phi(x)\frac{\partial S_w}{\partial t} + u_{tot}\frac{\partial f_w(S_w)}{\partial x} = 0, \tag{5}$$

and the total velocity u_{tot} is constant. After introducing the dimensionless variables $t_D = \int_0^t [(u_{tot}dt')/\phi L]$ and $x_D = x/L$, where L is the length of the one-dimensional system, we can rewrite Eq. (5) as follows:

$$\frac{\partial S_w}{\partial t_D} + \frac{\partial f_w(S_w)}{\partial x_D} = 0, \tag{6}$$

while initial and boundary conditions can be written as:

$$\begin{aligned} &S_w(x_D, 0) = s_{wi}, \quad \forall x_D \\ &S_w(x_D, t_D) = s_b, \quad x_D = 0 \quad \text{and} \quad t_D > 0. \end{aligned} \tag{7}$$

Solving this initial value problem is equivalent to solving the following nonlinear hyperbolic PDE:

$$\frac{\partial u}{\partial t} + \frac{\partial f(u)}{\partial x} = 0, \tag{8}$$

with the piecewise constant initial condition,

$$u(t = 0, x) = u^0(x). \tag{9}$$

Here, $u(t, x)$ is the space-time dependent quantity of interest (conserved scalar) that needs to be solved for, and $f(u)$ is the flux function. The PDE (8) can be solved by the method of characteristics, and it can be shown that the characteristics are straight lines [see e.g., (Lax, 1973)]. If the initial data (9) are piecewise constant having a single discontinuity, i.e., a Riemann problem, the PDE solution is a self-similar function. The hyperbolic PDE of the general form (8) is the main subject of the current work, and in the following we solve the initial value problem, Eqs. (8) and (9), by applying the physics informed machine learning (PIML) approach.

3. PHYSICS INFORMED MACHINE LEARNING

In this section we consider the following general partial differential equation:

$$u_t + \mathcal{N}(u) = 0, \tag{10}$$

where $\mathcal{N}(\cdot)$ is a nonlinear differential operator.

Neural networks are often regarded as universal function approximators (Cybenko, 1989; Hornik et al., 1989)—which means that a feed-forward network with a single hidden layer containing a finite number of neurons can approximate any continuous function to any desired level of precision. Following the approach of Raissi et al. (2019), the solution $u(t, x)$ to the PDE is approximated by a deep neural network parameterized by a set of parameters θ. In other words,

the solution to the PDE is represented as a series of function compositions:

$$\begin{aligned} y_1(t,x) &= \sigma(W_1 X + b_1), \\ y_2(t,x) &= \sigma(W_2 y_1 + b_2), \\ &\dots \\ y_{n_l+1}(t,x) &= W_{n_l+1} y_{n_l} + b_{n_l+1}, \\ u_\theta(t,x) &= y_{n_l+1}(t,x), \end{aligned} \tag{11}$$

where the input vector X contains space and time coordinates; i.e., $X = (x,t)$, θ is the ensemble of all the model parameters:

$$\theta = \{W_1, W_2, \dots, W_{n_l+1}, b_1, b_2, \dots, b_{n_l+1}\}, \tag{12}$$

σ is an activation function (tanh in our case) and n_l is the number of hidden layers. Defining $z_i(x) = \sigma(W_i x + b_i)$ for $i = 1, ..., n_l$ and $z_i(x) = W_i x + b_i$ for $i = n_l + 1$, we can write the solution to the PDE as follows:

$$u_\theta(t,x) = z_{n_l+1}(z_{n_l}(\dots z_2(z_1(X)))). \tag{13}$$

The residual of the PDE is just the left-hand side of Eq. (10):

$$r(t,x) = u_t + \mathcal{N}(u). \tag{14}$$

When the PDE solution is approximated by a neural network $u_\theta(t,x)$, the residual of the PDE can be also represented as the neural network with the same parameters θ:

$$r_\theta(t,x) = (u_\theta)_t + \mathcal{N}(u_\theta). \tag{15}$$

This network $r_\theta(t,x)$ can be easily derived by applying automatic differentiation to the network $u_\theta(t,x)$. Then, the shared parameters θ are learned by minimizing the following loss function:

$$\begin{aligned} L(\theta) &= L_u(\theta) + L_r(\theta), \\ L_u(\theta) &= \frac{1}{N_u}\sum_{i=1}^{N_u} |u_\theta(t_u^i, x_u^i) - u_{bc}^i|^2, \\ L_r(\theta) &= \frac{1}{N_r}\sum_{i=1}^{N_r} |r_\theta(t_r^i, x_r^i)|^2, \end{aligned} \tag{16}$$

where $\{(t_u^i, x_u^i), u_{bc}^i\}_{i=1}^{N_u}$ represent the training data on initial and boundary conditions, and $\{t_r^i, x_r^i\}_{i=1}^{N_r}$ denote the collocation points for the PDE residual, $r(t,x)$, sampled randomly throughout the domain of interest. Thus, the loss function consists of two terms: one is the mean squared error coming from the initial and boundary conditions, and the other is the mean squared error from the residual evaluated at collocation points inside the physical domain.

4. NUMERICAL RESULTS

In our examples, we consider the nonlinear hyperbolic transport equation of the form

$$u_t + (f_w)_x = 0, \tag{17}$$

where $f_w = f_w(u)$ is the fractional flow function, i.e., flux function, and $x \in [0,1], t \in [0,1]$. The unknown solution u corresponds to water saturation, S_w, in Eq. (6). Different flux functions produce different types of waves in the solution. In addition, we assume the following uniform initial and boundary conditions:

$$\begin{aligned} u(x,t) &= 0, \quad \forall x \quad \text{and} \quad t = 0, \\ u(x,t) &= 1, \quad x = 0 \quad \text{and} \quad t > 0. \end{aligned} \tag{18}$$

This setting corresponds to the injection of water at one end of the oil-filled 1D reservoir, e.g., rock core, and the following parameters: $s_{wi} = 0$, $s_b = 1$. The conservation law (17) with initial and boundary conditions (18) forms a Riemann problem that has a self-similar solution, i.e., $u(x,t) = u(x/t)$.

In the numerical examples, we use the fully connected neural network architecture reported in Raissi et al. (2019) that consists of eight hidden layers with 20 neurons per hidden layer. The hyperbolic tangent activation function is used in all hidden layers. All weights are initialized randomly according to the Xavier initialization scheme (Glorot and Bengio, 2010). The loss function is optimized with a second-order quasi-Newton method, L-BFGS-B (Nocedal and Wright, 2006). For the training data in all examples we use $N_u = 300$ randomly distributed points on initial and boundary conditions, and $N_r = 10{,}000$ collocation points for the residual term, sampled randomly over the interior of the domain $x \in [0,1], t \in [0,1]$. Next, we consider different flux functions $f_w(u)$ in Eq. (17).

4.1 Concave Flux Function

If the relative phase permeabilities, $k_{r\alpha}(S_\alpha)$, are linear functions of saturation, and the ratio of the phase viscosities is denoted as $\mu_o/\mu_w = M$, the corresponding flux function, f_w, can be written as

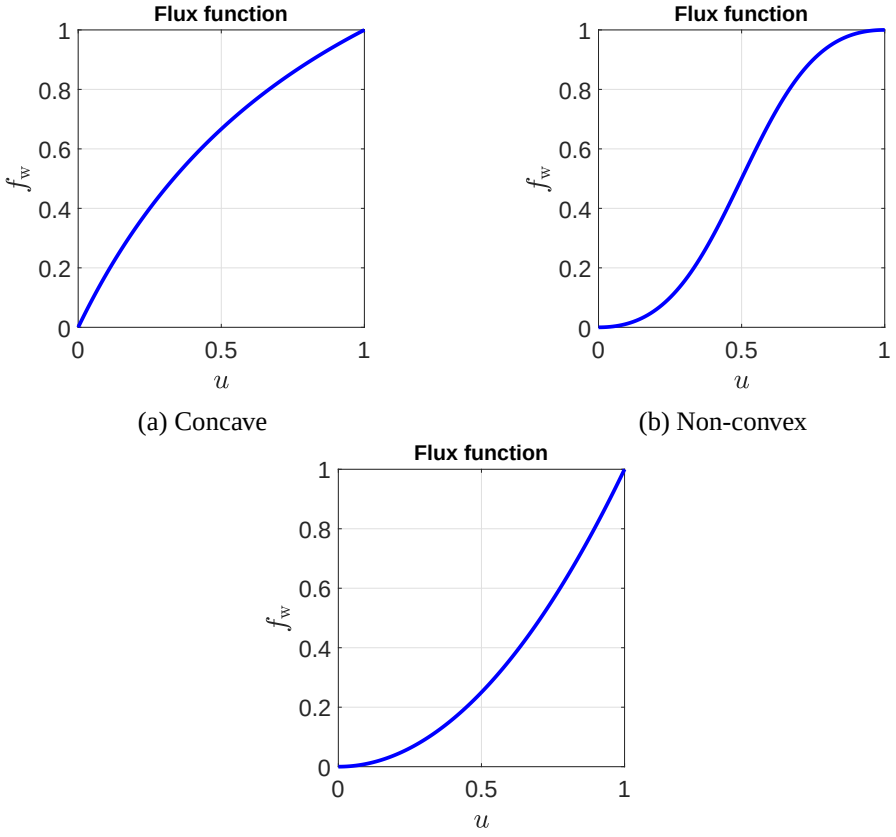
$$f_w(u) = \frac{u}{u + \dfrac{1-u}{M}}. \tag{19}$$

For $M > 1$, this flux function is concave, as shown in Fig. 1(a) for $M = 2$. The solution of the Eq. (17) for the given initial and boundary conditions (18) and the flux function (19) is a rarefaction (spreading) wave:

$$u(x,t) = \begin{cases} 0, & \dfrac{x}{t} > M \\ \dfrac{\sqrt{M\dfrac{t}{x}} - 1}{M-1}, & M \geq \dfrac{x}{t} \geq \dfrac{1}{M} \\ 1, & \dfrac{1}{M} \geq \dfrac{x}{t} \end{cases}.$$

We consider the case $M = 2$. Due to the piecewise nature of the analytical solution, there are certain locations (specifically, those along the lines $x/t = M$ and $x/t = 1/M$), where the solution is non-differentiable as derivatives of the solution are different on both sides.

However, this does not prevent the deep learning approach from learning the solution. Figure 2 presents a comparison of the exact analytical solution and the solution predicted by neural network at time instances $t = 0.25, 0.5, 0.75$. In this case, the neural network produces accurate



(a) Concave (b) Non-convex

(c) Convex

FIG. 1: Different flux functions

estimates of the PDE solution with some smoothing of the non-differentiable edges of the solution. The final loss at the end of training is $L(\theta) = 1.2 \times 10^{-3}$ and the resulting relative $\mathcal{L}^2$ norm of the prediction error of the solution (compared to the analytical solution) is 2.6×10^{-2}.

4.2 Non-Convex Flux Function

In most practical settings, the interaction between two immiscible fluids flowing through the porous medium leads to highly nonlinear relative permeabilities. A simple model that captures this characteristic is the Brooks-Corey model (Brooks and Corey, 1964), which gives the power-law relationship between the relative permeability of a fluid phase and its saturation. Specifically, we use a quadratic relationship, which leads to the following flux, i.e., fractional flow, function:

$$f_w(u) = \frac{u^2}{u^2 + \frac{(1-u)^2}{M}}, \tag{20}$$

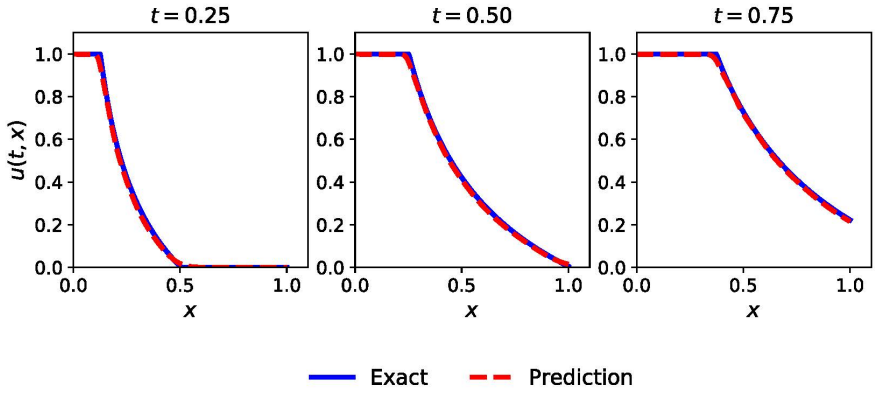


FIG. 2: Comparison of the predicted by the neural network and the exact solutions of the PDE (17) with concave flux function (19), corresponding to the three different times $t = 0.25, 0.5, 0.75$

where again M is the ratio of phase viscosities. The PDE (17) with this non-convex flux function constitutes a standard Buckley-Leverett problem in porous media flow. In our example we use $M = 1$ and the corresponding flux function is depicted in Fig. 1(b). We proceed by considering two cases with this flux function—with and without an additional diffusion term in the PDE.

4.2.1 Without Diffusion Term

In this case, the residual term (17), representing the hyperbolic PDE, is used directly in the loss function. The analytical solution to this problem contains a shock and a rarefaction wave and is constructed as follows:

$$u(x,t) = \begin{cases} 0, & \frac{x}{t} > f'_w(u^*) \\ u\left(\frac{x}{t}\right), & f'_w(u^*) \geq \frac{x}{t} \geq f'_w(u=1) \\ 1, & f'_w(u=1) \geq \frac{x}{t} \end{cases}, \tag{21}$$

where u^* denotes the shock location, which is defined by the Rankine-Hugoniot condition $f'_w(u^*) = [f_w(u^*) - f_w(u)_{|u=0}]/(u^* - u_{|u=0})$, and $u(x/t)$ is defined for $x/t \leq f'_w(u^*)$ as $u(x/t) = (f'_w)^{-1}(x/t)$. Due to the self-similarity, the analytical solution (21) has just one governing parameter—the similarity variable x/t.

Figure 3 shows that the neural network fails in this case to provide an accurate approximation of the underlying analytical solution (21). In fact, the neural network completely misses the correct location of the saturation front, which leads to high values of the loss [at the end of training it is $L(\theta) = 0.036$] and large prediction errors. In our numerical experiments, we observed that changing the neural network architecture and/or increasing the number of collocation points had little impact on the results (details of these studies are provided in Appendix A). Thus, we think this phenomenon is not related to the choice of the network architecture or its hyperparameters.

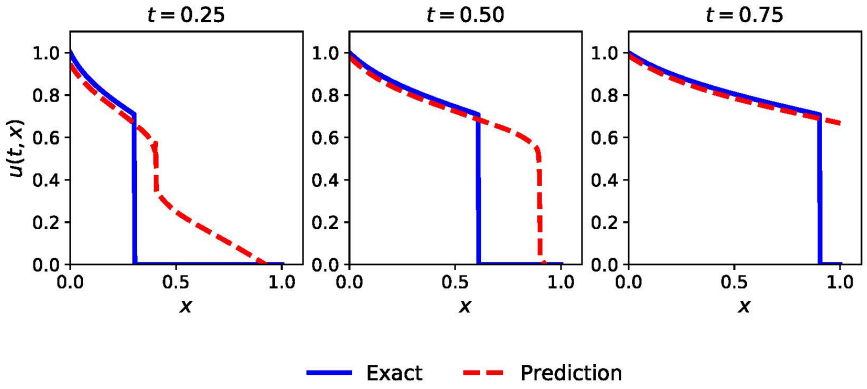


FIG. 3: Comparison of the predicted by the neural network and the exact solutions of the PDE (17) with non-convex flux function (20), corresponding to the three different times $t = 0.25, 0.5, 0.75$

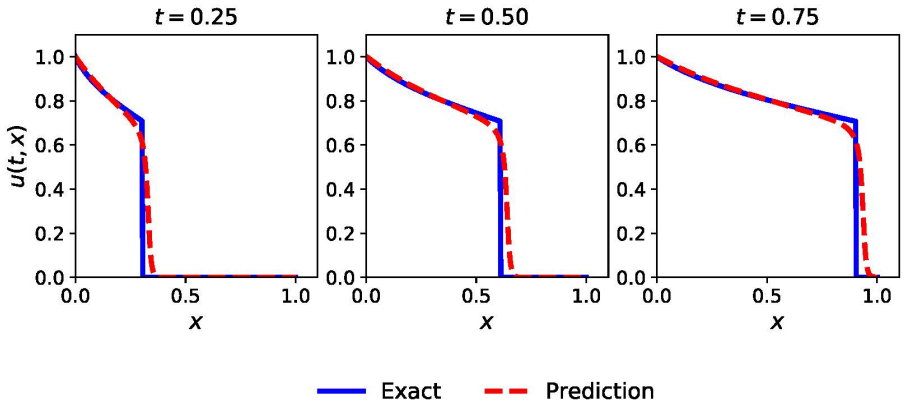
4.2.2 With Diffusion Term

The vanishing viscosity method for solving the initial value problems for hyperbolic PDEs (Crandall and Lions, 1983; Lax, 2006) is based on the fact that solutions of the inviscid equations, e.g., Eq. (17), including solutions with shocks, are the limits of the solutions of the viscous equations as the coefficient of viscosity tends to zero. Motivated by this approach, we add a second-order term, i.e., a diffusion term, to the right-hand side of Eq. (17) and consider the following equation:

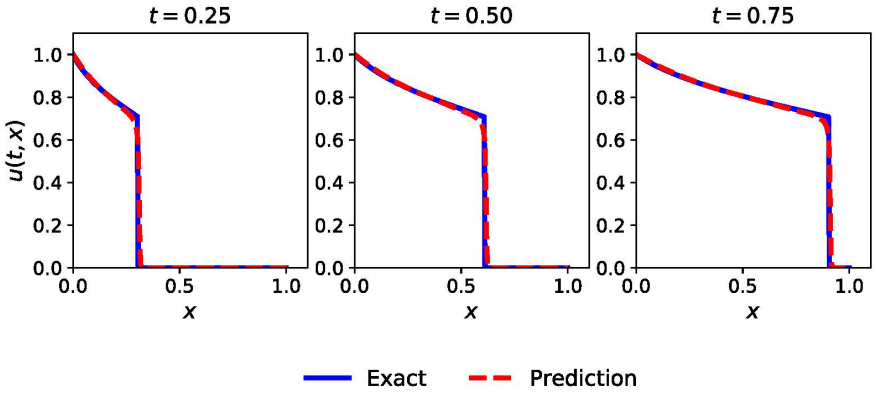
$$u_t + f'_w(u)u_x = \epsilon u_{xx}, \tag{22}$$

where $\epsilon > 0$ is a scalar diffusion coefficient that represents the inverse of the Péclet number, Pe—the ratio of a characteristic time for dispersion to a characteristic time for convection. When ϵ is small, i.e., the Péclet number is large, the effects of diffusion are negligible and convection dominates. Letting $\epsilon \to 0$ in Eq. (22) defines a vanishing diffusion solution of Eq. (17), which is the one with the correct physical behavior. Also, it should be noted that Eq. (22) is now a parabolic PDE, so its solution is smooth, i.e., it does not contain shocks.

Figure 4 shows neural network solutions for two different values of diffusion coefficient ϵ: 1×10^{-2} (Pe $= 100$) and 2.5×10^{-3} (Pe $= 400$). The loss values at the end of the training are $L(\theta) = 3.2 \times 10^{-6}$ and 2.4×10^{-5}, respectively. Note that the loss function is different in these two cases as the loss depends on the PDE residual, which is a function of ϵ according to Eq. (22). From these results, we see that adding a diffusion term to the conservation equation allows the neural network to perfectly capture the location of the saturation front even for quite small ϵ. Indeed, the solution in Fig. 4(b) for $\epsilon = 2.5 \times 10^{-3}$ is almost indistinguishable from the underlying analytical PDE solution—there is just a slight smoothing of the shock. In our numerical experiments, we also observed that if we continue to decrease the value of diffusion coefficient ϵ, e.g., $\epsilon = 1 \times 10^{-3}$, then the diffusion effects become too small, and the behavior of the neural network is the same as in the hyperbolic setting (i.e., zero diffusion) described in Section 4.2.1. It should be noted that the experiments in the current section—both for PDEs with and without the diffusion term—were all performed multiple times with different random seeds and random initializations; however, the results in terms of recovering the shock were equivalent.



(a) $\epsilon = 1 \times 10^{-2}$



(b) $\epsilon = 2.5 \times 10^{-3}$

FIG. 4: Predictions of the neural network for the PDE (22) for different values of diffusion coefficient ϵ. Exact solution corresponds to the PDE (17) without diffusion term.

Then, we conducted similar experiments for other values of phase viscosity ratio M, such as $M = 0.5, 5, 10$, that are also common in the subsurface transport domain. Under these settings the solutions differ in size and speed of the shock, i.e., for larger M the shock size decreases but its speed increases. However, the solution structure stays exactly the same—the solution still consists of a shock followed by a rarefaction wave. We considered also two cases for each value of the phase viscosity ratio M—with and without the diffusion term. The results of these tests and conclusions were the same as for $M = 1$ described above; thus we conclude that the observed behavior of the PIML approach is not sensitive to the value of parameter M.

It is worth mentioning that the obtained results are consistent with the previously reported results of the PIML approach in Raissi et al. (2017). The authors of Raissi et al. (2017) studied Burgers' equation with the diffusion term (so the shock was smoothed) and the diffusion coefficient (ϵ in our notation) was equal to $\epsilon = 0.01/\pi \approx 3.2 \times 10^{-3}$. However, if one applies the PIML approach for the same settings of Burgers' equation as in Raissi et al. (2017) but decreases

the diffusion coefficient to 0.5×10^{-3} or less (or sets it to zero altogether), then the network fails in a similar way as was described in Section 4.2.1.

4.3 Convex Flux Function

Now, we move to the convex flux function, shown in Fig. 1(c), which is simply a quadratic function $f_w(u) = u^2$. The solution is a self-sharpening wave, propagating as shock with a unit speed.

The prediction of the neural network for $t = 0.5$ in the case of hyperbolic PDE (17) is shown in the left plot of Fig. 5. As in the case of the non-convex flux function, the PIML approach fails for this problem. And similar to the non-convex flux case, adding a small diffusion term, e.g., with $\epsilon = 2.5 \times 10^{-3}$, to the PDE allows the neural network to reconstruct the solution and determine the location of the (smoothed) shock correctly (Fig. 5, on the right).

5. ANALYSIS

It is quite surprising that the neural network with several thousands of parameters is not able to yield a reasonable approximation to the analytical solution of the 1D hyperbolic PDE (17) with a non-convex flux function (20)—the solution that can be represented using a relatively simple piecewise continuous function of one parameter (21). This is surprising, especially because according to the universal approximation theorem (Cybenko, 1989) there should exist a network that can provide a close approximation of the continuous solution of (22) for any arbitrarily small ϵ (because the solution is smooth in this case); however, this is not what is observed in practice. Thus, this leads us to the conclusion that the problem is not with the solution itself, but rather with *how* we attempt to find this solution, i.e., with the optimization process, or the loss function.

For the examples described above, we provide the analysis of the obtained neural networks. Our aim here is to get a better understanding of the observed behavior of the neural network approach—why it can find a solution to the problem with the additional diffusion term, i.e., the parabolic form of the PDE, but fails to do so in the case of the underlying hyperbolic PDE, i.e., when its solution contains a discontinuity. Is this due to some fundamental reasons that prevent the neural network from finding a reasonable approximate solution (non-uniqueness of

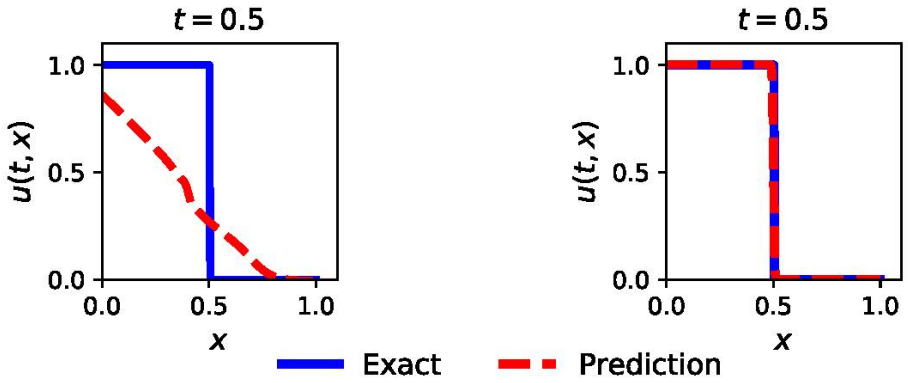


FIG. 5: Predictions of the neural network at $t = 0.5$ for the case of convex flux function: on the left the prediction for the PDE without diffusion term, on the right – with added diffusion term, as in Eq. (22), and diffusion coefficient $\epsilon = 2.5 \times 10^{-3}$. Exact solutions in both cases are shown for the PDE (17) without diffusion term.

the solution of the weak form), or is it because the employed optimization algorithm just cannot reach the solution? The latter can be due to the complicated nature of the non-convex landscape of the loss function, or other inherent limitations of the optimization algorithm.

First, we investigate the training process and study the behavior of the loss and its gradients with respect to the network parameters. Then, through 2D visualizations of the loss surface, we study how the diffusion term affects the loss landscape and the convexity of the loss near the final optimization point, i.e., optimized set of network parameters.

5.1 Training Process

Figure 6 shows the evolution of the loss function during the training process for models with different amounts of diffusion, i.e., different values of the diffusion coefficient ϵ before the second-order term in Eq. (22). The x-axis in the figure denotes the steps of the L-BFGS-B optimization method. Note that the loss function being minimized is different for each model, as part of the loss, corresponding to the residual term, is directly proportional to ϵ. In Fig. 6 we observe a clear trend. For larger values of ϵ the convergence rate of the optimization improves significantly, i.e., the loss is minimized in far fewer steps. On the other hand, for smaller values (i.e., $\epsilon = 0$ or 1×10^{-3}) the corresponding loss curve flattens out quite early during the training, and the optimization method fails to minimize the loss (the final loss is only of order 10^{-2}).

The training of the neural network can also be studied by observing the gradient of the loss with respect to the different parameters of the network, i.e., weights and biases of different layers. Figure 7 shows the $\mathcal{L}^2$ norm of the loss gradient with respect to the weights in the first layer versus the number of optimization steps (some curves were smoothed for better visualization). The curves for the models that achieve good approximation accuracy of the solution, i.e., the models with $\epsilon = 5 \times 10^{-2}$, 5×10^{-3} and 2.5×10^{-3}, show a steady decrease in the norm

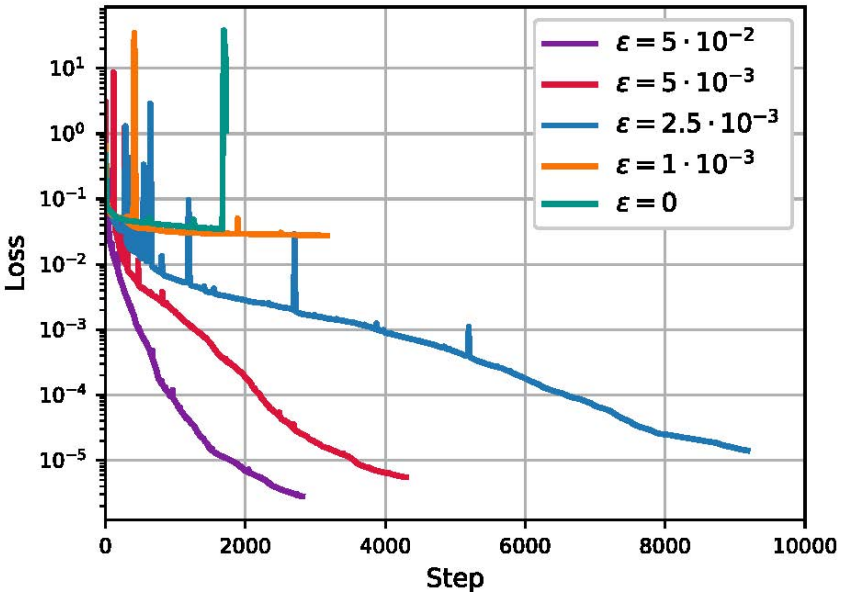


FIG. 6: The loss function during training for models with different amount of added diffusion according to Eq. (22). The x-axis denotes the steps of the L-BFGS-B optimization method.

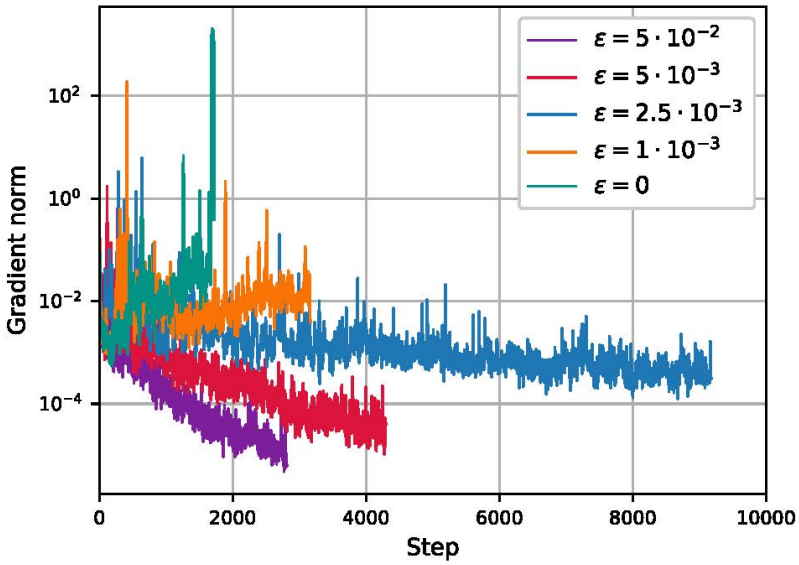


FIG. 7: The evolution of the $\mathcal{L}^2$ norm of the loss gradient with respect to the weights of the first network layer during training. The x-axis denotes the steps of the L-BFGS-B optimization method.

of the gradient during training, indicating convergence of the optimization process; on the other hand, for the models that have large prediction errors, i.e., for $\epsilon = 0$ and $\epsilon = 1 \times 10^{-3}$, the gradients do not decrease with time, and sometimes even increase, indicating failure of the optimization process. The loss gradients with respect to the parameters in other layers of the network showed similar trends. Again, from the results shown in Fig. 7 it is obvious that the magnitude of ϵ significantly affects the behavior of the loss gradients. This behavior for $\epsilon \sim 0$ may be explained with the complicated objective function landscape, so that the quasi-Newton method fails to minimize the loss. It may also be due to the poor conditioning of the Hessian of the loss, so that the desired solution lies in a very local and narrow region. Nevertheless, it is clear that the presence of the second-order term u_{xx}, i.e., presence of diffusion in the PDE, and the amount of diffusion strongly influence the training process of the physics informed network and its ability to yield accurate approximations of the solution.

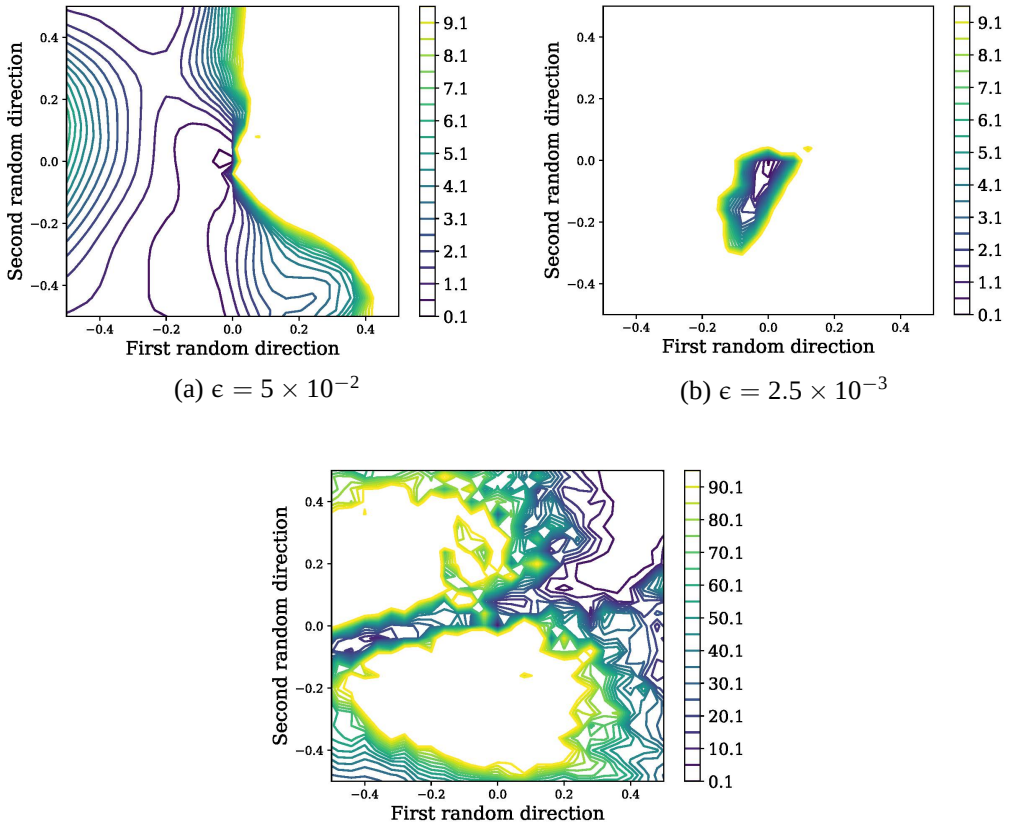
5.2 Loss Landscape

To visualize the surface of the loss, which is a function in the high-dimensional parameter space, one must restrict the space to a low-dimensional one (1D or 2D), amenable to visualization. Here, we choose to follow the approach of Li et al. (2018), whereby to get a 2D projection of the loss surface we choose a center point θ, corresponding to the final optimization point (i.e., final parameters of the model reshaped into a single vector) and two direction vectors, δ and η, of the same dimension as θ. Then, we can plot the following function:

$$f(\alpha, \beta) = L(\theta + \alpha\delta + \beta\eta), \tag{23}$$

where α and β are scalar parameters along vectors δ and η, respectively. The direction vectors are sampled randomly from Gaussian distribution—in the high-dimensional space these vectors with a high probability will be almost orthogonal to each other. Then, Li et al. (2018) suggest "filter-wise" normalizing the random directions to capture the natural distance scale of the loss surface. This step ensures that elements in random vectors, δ and η, are of the same scale as the corresponding parameters of the network, i.e., weights and biases of different network layers.

For visualizations we vary both scalar parameters, α and β, in the range $(-0.5, 0.5)$. Figure 8 shows the loss surface plots for different networks near their final optimization point, i.e., set of optimized parameters. This point corresponds to $(0, 0)$ in the surface plots, and the two axes represent the two random directions, respectively. The results are shown as contour plots to make it easier to see the non-convex structures of the loss landscape. The networks differ in the amount of the added diffusion, i.e., value of diffusion coefficient ϵ. For large diffusion, for example, $\epsilon = 5 \times 10^{-2}$, in Fig. 8(a), we observe quite a large convex region, whereas for a small amount of diffusion, e.g., $\epsilon = 2.5 \times 10^{-3}$, this region shrinks significantly, as shown in



(a) $\epsilon = 5 \times 10^{-2}$

(b) $\epsilon = 2.5 \times 10^{-3}$

(c) $\epsilon = 0$

FIG. 8: 2D visualizations of the loss surface near the final optimization point for neural networks trained with different values of diffusion coefficient ϵ in Eq. (22). Note the change of scale for $\epsilon = 0$.

Fig. 8(b). Note the change of scale in Fig. 8(c), which depicts the loss surface for the hyperbolic PDE, i.e., $\epsilon = 0$,—for proper visualization the scale of the loss had to be increased by 10 times compared to the cases with diffusion. No convex region is observed in this instance. Moreover, the loss landscape is not as smooth as with the diffusion present—indeed, it has a lot of chaotic features, as can be seen in Fig. 8(c). For visualization of the same loss surface on a larger slice of the parameter space, refer to Fig. 9. From these observations, we can conclude that the presence of the discontinuity, i.e., the shock, in the PDE solution strongly affects the properties of the resulting landscape of the corresponding loss function – specifically, its smoothness and convexity. It is not surprising that the optimization procedure struggles with this loss landscape and is unable to reach the proper solution, i.e., the one that gives a close continuous approximation of the discontinuous PDE solution (21). For comparison, we also show in Fig. 10 the loss surface of the network approximating a smooth PDE solution in case of concave flux function (19). The wide convex region of the loss surface is evident here.

6. DISCUSSION AND CONCLUSION

We investigated the application of a physics informed machine learning (PIML) approach to the solution of one-dimensional hyperbolic PDEs that describe the nonlinear two-phase transport in porous media. The PIML approach encodes the underlying PDE into the loss function and learns the solution to the PDE without any labeled data—only using the knowledge of the initial/boundary conditions and the PDE. Our experiments with different flux functions demonstrate that the neural network approach provides accurate estimates of the solution of the hyperbolic PDE when the solution does not contain discontinuities. However, the PIML approach fails to provide reasonable approximate solution of the PDE when shocks are present. We found that it is necessary to add a diffusion term to the underlying PDE, so that the network can recover the proper location and size of the shock, which is smoothed by diffusion. Thus, the network actually solves the parabolic form of the conservation equation, which leads to the correct solution with smoothing around the shock. It is interesting to note here the resemblance of this effect with finite-volume methods, whereby the conservative finite-volume discretization adds a

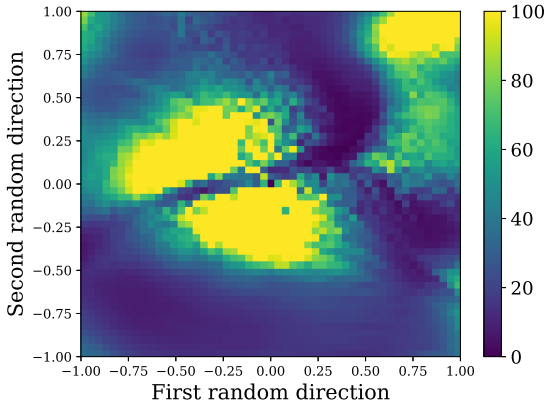


FIG. 9: 2D visualization of the loss surface of the neural network for the hyperbolic PDE (17) with non-convex flux function (20)

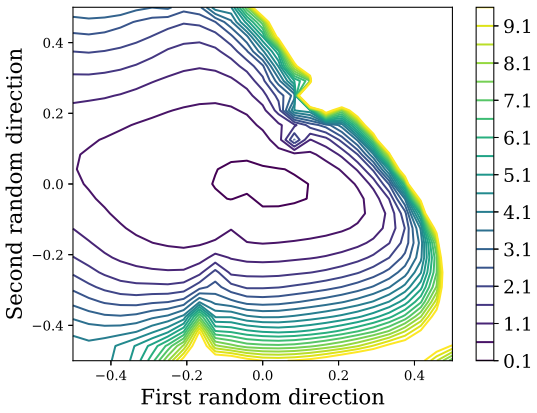


FIG. 10: 2D visualization of the loss surface of the neural network for the hyperbolic PDE (17) with concave flux function (19) (the PDE solution is smooth in this case)

numerical diffusion term, and as a result, the numerical solution corresponds to a parabolic equation with a finite amount of diffusion. This diffusion term can be controlled through refinement in space-time and by the use of higher-order discretization schemes.

Then, we analyzed the network training process for cases with and without diffusion in the PDE. Our study shows that the amount of added diffusion strongly affects the training of the network (e.g., the convergence rate, the behavior of the loss gradients). Moreover, we provided 2D visualizations of the loss landscape of the neural networks near their final optimization point, which indicate that the diffusion term in the PDE smooths the loss surface and makes it more convex, while the loss surface of the hyperbolic PDE with discontinuous solution demonstrates significant chaotic and non-convex features. However, the reasons for such behavior of the loss function are not perfectly understood yet. It would be certainly interesting to derive some analytical explanation of the observed phenomena as well. Nevertheless, through the experiments and analysis conducted in the current work we show that the physics informed machine learning framework is not suited for the hyperbolic PDEs with discontinuous solutions considered here.

ACKNOWLEDGMENTS

We thank Total for their financial support of our research on “Uncertainty Quantification.” The authors are also grateful to the Stanford University Petroleum Research Institute for Reservoir Simulation (SUPRI-B) for financial support of this work.

REFERENCES

Aziz, K. and Settari, A., *Petroleum Reservoir Simulation*, London: Elsevier/8, Applied Science Publishers, 1979.

Brooks, R. and Corey, T., Hydraulic Properties of Porous Media, *Hydrology Papers, Colorado State University*, vol. **24**, 1964.

Crandall, M.G. and Lions, P.L., Viscosity Solutions of Hamilton-Jacobi Equations, *Trans. Am. Math. Soc.*, vol. **277**, no. 1, pp. 1–42, 1983.

Cybenko, G., Approximation by Superpositions of a Sigmoidal Function, *Math. Control, Signals Syst.*, vol. **2**, no. 4, pp. 303–314, 1989.

Glorot, X. and Bengio, Y., Understanding the Difficulty of Training Deep Feedforward Neural Networks, in *Proc. of the 13th Int. Conf. on Artificial Intelligence and Statistics*, pp. 249–256, 2010.

Gulshan, V., Peng, L., Coram, M., Stumpe, M.C., Wu, D., Narayanaswamy, A., Venugopalan, S., Widner, K., Madams, T., Cuadros, J., Kim, R., Raman, R., Nelson, P.C., Mega, J.L., and Webster, D.R., Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs, *J. Am. Med. Assoc.*, vol. **316**, no. 22, pp. 2402–2410, 2016.

He, K., Zhang, X., Ren, S., and Sun, J., Deep Residual Learning for Image Recognition, in *Proc. of the IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 770–778, 2016.

Hinton, G., Deng, L., Yu, D., Dahl, G.E., Mohamed, A.R., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., Sainath, T.N., and Kingsbury, B., Deep Neural Networks for Acoustic Modeling in Speech Recognition, *IEEE Signal Process. Mag.*, vol. **29**, no. 6, pp. 82–97, 2012.

Hornik, K., Stinchcombe, M., and White, H., Multilayer Feedforward Networks are Universal Approximators, *Neural Networks*, vol. **2**, no. 5, pp. 359–366, 1989.

Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., and Fei-Fei, L., Large-Scale Video Classification with Convolutional Neural Networks, *Proc. of the 2014 IEEE Conf. on Computer Vision and Pattern Recognition*, pp. 1725–1732, 2014.

Krizhevsky, A., Sutskever, I., and Hinton, G.E., Imagenet Classification with Deep Convolutional Neural Networks, in *Adv. Neural Inf. Process. Syst.*, vol. **25**, no. 2, pp. 1097–1105, 2012.

Lagaris, I.E., Likas, A., and Fotiadis, D.I., Artificial Neural Networks for Solving Ordinary and Partial Differential Equations, *IEEE Trans. Neural Networks*, vol. **9**, no. 5, pp. 987–1000, 1998.

Lax, P.D., *Hyperbolic Systems of Conservation Laws and the Mathematical Theory of Shock Waves*, Philadelphia: Society for Industrial and Applied Mathematics, vol. **11**, 1973.

Lax, P.D., *Hyperbolic Partial Differential Equations*, Providence, RI: American Mathematical Soc., vol. **14**, 2006.

LeCun, Y. and Bengio, Y., Convolutional Networks for Images, Speech, and Time Series, in *The Handbook of Brain Theory and Neural Networks*, M.A. Arbib, Ed., Cambridge MA: The MIT Press, vol. **3361**, no. 10, 1995.

LeCun, Y., Bengio, Y., and Hinton, G., Deep Learning, *Nature*, vol. **521**, no. 7553, pp. 436–444, 2015.

Lee, H. and Kang, I.S., Neural Algorithm for Solving Differential Equations, *J. Comput. Phys.*, vol. **91**, no. 1, pp. 110–131, 1990.

Li, H., Xu, Z., Taylor, G., Studer, C., and Goldstein, T., Visualizing the Loss Landscape of Neural Nets, *Adv. Neural Inf. Process. Syst.*, pp. 6389–6399, 2018.

Lillicrap, T.P., Hunt, J.J., Pritzel, A., Heess, N., Erez, T., Tassa, Y., Silver, D., and Wierstra, D., Continuous Control with Deep Reinforcement Learning,, 2015. arXiv: 1509.02971

Liu, Y., Gadepalli, K., Norouzi, M., Dahl, G.E., Kohlberger, T., Boyko, A., Venugopalan, S., Timofeev, A., Nelson, P.Q., and Corrado, G.S., Detecting Cancer Metastases on Gigapixel Pathology Images, 2017. arXiv: 1703.02442

Meade Jr., A.J. and Fernandez, A.A., The Numerical Solution of Linear Ordinary Differential Equations by Feedforward Neural Networks, *Math. Comput. Model.*, vol. **19**, no. 12, pp. 1–25, 1994.

Mnih, V., Badia, A.P., Mirza, M., Graves, A., Lillicrap, T., Harley, T., Silver, D., and Kavukcuoglu, K., Asynchronous Methods for Deep Reinforcement Learning, in *Proc. of the 33rd Int. Conf. on Machine Learning*, pp. 1928–1937, 2016.

Nocedal, J. and Wright, S., *Numerical Optimization*, Berlin: Springer Science & Business Media, 2006.

Orr, F., *Theory of Gas Injection Processes*, Holte, Denmark: Tie-Line Publications, 2007.

Psichogios, D.C. and Ungar, L.H., A Hybrid Neural Network-First Principles Approach to Process Modeling, *AIChE J.*, vol. **38**, no. 10, pp. 1499–1511, 1992.

Raissi, M., Perdikaris, P., and Karniadakis, G.E., Physics Informed Deep Learning (Part I): Data-Driven Solutions of Nonlinear Partial Differential Equations, 2017. arXiv: 1711.10566

Raissi, M., Perdikaris, P., and Karniadakis, G.E., Physics-Informed Neural Networks: A Deep Learning Framework for Solving Forward and Inverse Problems Involving Nonlinear Partial Differential Equations, *J. Comput. Phys.*, vol. **378**, pp. 686–707, 2019.

Stewart, R. and Ermon, S., Label-Free Supervision of Neural Networks with Physics and Domain Knowledge, in *Proc. of the 31st AAAI Conference on Artificial Intelligence*, pp. 2576–2582, 2017.

Sutskever, I., Vinyals, O., and Le, Q.V., Sequence to Sequence Learning with Neural Networks, *Adv. Neural Inf. Process. Syst.*, pp. 3104–3112, 2014.

Zhu, Y., Zabaras, N., Koutsourelakis, P.S., and Perdikaris, P., Physics-Constrained Deep Learning for High-Dimensional Surrogate Modeling and Uncertainty Quantification without Labeled Data, *J. Comput. Phys.*, vol. **394**, pp. 56–81, 2019.

APPENDIX A. SENSITIVITY STUDY FOR THE BUCKLEY-LEVERETT PROBLEM

For the case described in Section 4.2.1, we perform a sensitivity study. Our aim here is to understand whether the result obtained in Section 4.2.1 for the Buckley-Leverett problem (nonlinear transport with a non-convex flux function) is strongly dependent on the particular choice of the network architecture and the different hyperparameters of the method, such as the number of training points in initial and boundary data N_u and the number of collocation points N_r in the interior of the domain.

First, we fix the network architecture to eight hidden layers with 20 neurons per hidden layer, and we vary the number of initial and boundary training data N_u in the range $(100, 600)$ and the number of collocation points N_r in the range (1000, 20,000). The final values of the loss function at the end of the training for these experiments are shown in Table A1. In all these cases, the network failed to yield a reasonable approximation of the shock; as the result, we observe a relatively large value of the loss function (i.e., $\sim 10^{-2}$). From Table A1, it is also clear that the network performance is not a strong function of the number of initial and boundary training data and the number of collocation points.

In the next experimental set, we kept the total number of training and collocation points fixed to $N_u = 300$ and $N_r = 10{,}000$, and varied the number of hidden layers in the range $(2, 12)$ and the number of neurons per hidden layer in the range $(10, 40)$. With these ranges, the total number of network parameters varied from 151 to over 18,000. Table A2 reports the value of the loss function at the end of the training for these different architectures. Again,

TABLE A1: Final loss at the end of training for different number of initial and boundary training data points N_u and different number of collocation points N_r. The network architecture is fixed to 8 hidden layers with 20 neurons per hidden layer

N_u \ N_r	**1000**	**5000**	**10,000**	**20,000**
100	1.6×10^{-2}	3.4×10^{-2}	3.0×10^{-2}	2.6×10^{-2}
300	2.2×10^{-2}	2.6×10^{-2}	3.4×10^{-2}	3.2×10^{-2}
600	1.3×10^{-2}	2.0×10^{-2}	3.1×10^{-2}	3.0×10^{-2}

TABLE A2: Final loss at the end of training for different number of hidden layers and different number of neurons per hidden layer. The total number of training and collocation points is fixed to $N_u = 300$ and $N_r = 10{,}000$

Layers \ Neurons	10	20	40
2	3.4×10^{-2}	3.2×10^{-2}	3.2×10^{-2}
4	1.6×10^{-2}	3.2×10^{-2}	3.1×10^{-2}
8	3.3×10^{-2}	3.4×10^{-2}	3.3×10^{-2}
12	3.5×10^{-2}	2.9×10^{-2}	1.9×10^{-2}

the observed trend is quite consistent—the final result is not weakly sensitive to the particular network architecture. Moreover, the PDE solutions $u(t, x)$ predicted by the neural networks in all these cases were quite similar to the ones reported in Section 4.2.1, where the network completely fails to approximate the shock.

In addition, we experimented with application of standard regularization of the network weights—the technique typically used in machine learning to decrease overfitting. Specifically, we added to the loss function $L(\theta)$ a regularization term of the form $l_{\text{reg}} = \beta W^T W$ (where W denotes the weights of the network) and considered a range of regularization constants $\beta = [1 \times 10^{-5}, 5 \times 10^{-5}, 1 \times 10^{-4}, 5 \times 10^{-4}, 1 \times 10^{-3}]$. However, in these experiments we did not see any improvement in the PIML results for hyperbolic PDE.

Next, we also tested the PIML approach with different types of networks—a residual network architecture (He et al., 2016) and a convolutional neural network (CNN) (LeCun et al., 1995). For the residual network we added skip connections after each layer in the original fully connected architecture. With CNN architecture we used eight convolutional layers with 20 filters each, that perform 1D convolutions and have a kernel size of 1×1 (in this case, the number of parameters is the same as in the standard fully connected architecture reported in the paper). In these experiments we observed similar behavior—that the PIML approach fails for hyperbolic PDE but performs well for PDE with added diffusion term.

TRAINABILITY OF ReLU NETWORKS AND DATA-DEPENDENT INITIALIZATION

Yeonjong Shin & George Em Karniadakis*

Division of Applied Mathematics, Brown University, Providence, Rhode Island 02912, USA

*Address all correspondence to: Yeonjong Shin, Division of Applied Mathematics, Brown University, Providence, Rhode Island 02912, USA; Tel.: +1 401 863 1320; Fax: +1 401 863 1355, E-mail: yeonjong_shin@brown.edu

Original Manuscript Submitted: 3/9/2020; Final Draft Received: 6/19/2020

In this paper we study the trainability of rectified linear unit (ReLU) networks at initialization. A ReLU neuron is said to be dead if it only outputs a constant for any input. Two death states of neurons are introduced—tentative and permanent death. A network is then said to be trainable if the number of permanently dead neurons is sufficiently small for a learning task. We refer to the probability of a randomly initialized network being trainable as trainability. We show that a network being trainable is a necessary condition for successful training, and the trainability serves as an upper bound of training success rates. In order to quantify the trainability, we study the probability distribution of the number of active neurons at initialization. In many applications, overspecified or overparameterized neural networks are successfully employed and shown to be trained effectively. With the notion of trainability, we show that overparameterization is both a necessary and a sufficient condition for achieving a zero training loss. Furthermore, we propose a data-dependent initialization method in an overparameterized setting. Numerical examples are provided to demonstrate the effectiveness of the method and our theoretical findings.

KEY WORDS: *ReLU networks, trainability, dying ReLU, overparameterization, overspecification, data-dependent initialization*

1. INTRODUCTION

Neural networks have been successfully used in various fields of applications. These include image classification in computer vision (Krizhevsky et al., 2012), speech recognition (Hinton et al., 2012), natural language translation (Wu et al., 2016), and superhuman performance in the game of Go (Silver et al., 2016). Modern neural networks are often severely overparameterized or overspecified. Overparameterization means that the number of parameters is much larger than the number of training data. Overspecification means that the number of neurons in a network is much larger than needed. It has been reported that the wider the neural networks, the easier it is to train (Livni et al., 2014; Nguyen and Hein, 2017; Safran and Shamir, 2016).

In general, neural networks are trained by first- or second-order gradient-based optimization methods from random initialization. Almost all gradient-based optimization methods stem from backpropagation (Rumelhart et al., 1985) and the stochastic gradient descent (SGD) method (Robbins and Monro, 1951). Many variants of vanilla SGD have been proposed; for example,

2689–3967/20/$35.00

AdaGrad (Duchi et al., 2011), RMSProp (Hinton, 2014), Adam (Kingma and Ba, 2015), AMSGrad (Reddi et al., 2019), and L-BFGS (Byrd et al., 1995), to name just a few. Different optimization methods have different convergence properties. It is still far from clear how different optimization methods affect the performance of trained neural networks. Nonetheless, how to start the optimization processes plays a crucial role for the success of training. Properly chosen weight initialization could drastically improve the training performance and allow the training of deep neural networks; for example, see LeCun et al. (1998), Glorot and Bengio (2010), Saxe et al. (2014), He et al. (2015), Mishkin and Matas (2016), and for more recent work see Lu et al. (2019). Among them, when it comes to the rectified linear unit (ReLU) neural networks, the "He initialization" (He et al., 2015) is one of the most commonly used initialization methods.

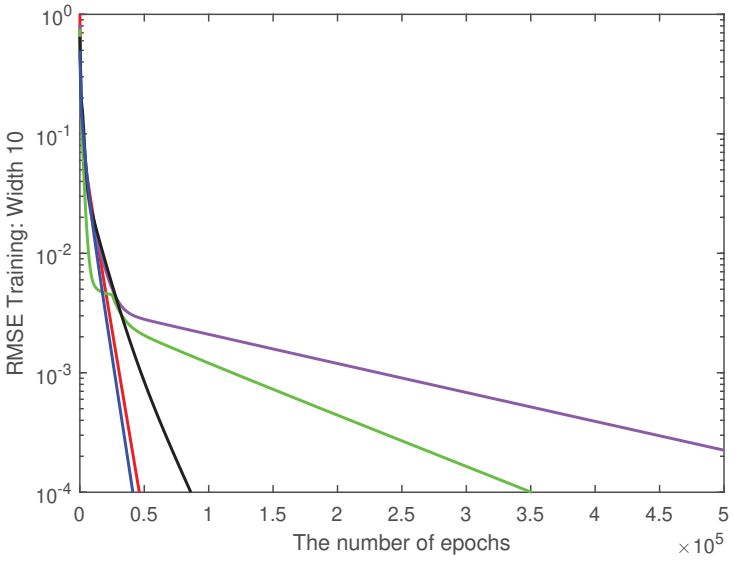
There are several theoretical works showing that under various assumptions, overparameterized neural networks can perfectly interpolate the training data. For the shallow (two-layer) neural network setting, see Oymak and Soltanolkotabi (2019), Soltanolkotabi et al. (2019), Du et al. (2018b), and Li and Liang (2018). For the deep (more than two layers) neural network setting, see Du et al. (2018a), Zou et al. (2018), and Allen-Zhu et al. (2018). Hence, overparameterization can be viewed as a sufficient condition for minimizing the training loss. In spite of the current theoretical progress, there still exists a huge gap between existing theories and empirical observations in terms of the level of overparameterization. To illustrate this gap, let us consider the problem of approximating $f(x) = |x|$. The same learning task was also used in Lu et al. (2019) but with a deep network. Here we consider a two-layer (shallow) rectified linear unit (ReLU) network. The training set consists of 10 random samples from the uniform distribution on $[-1, 1]$. To interpolate all 10 data points, the best existing theoretical condition requires the width of $\mathcal{O}(n^2)$ (Oymak and Soltanolkotabi, 2019). In this case, the width of 100 would be needed. Figure 1 shows the convergence of the root-mean-square errors (RMSE) on the training data with respect to the number of epochs for five independent simulations. On the left, the results of width 10 are shown. We observe that all five training losses converge to zero as the number of epochs increases. It would be an ongoing challenge to bridge the gap of the degree of overparameterization.

On the other hand, we know that $f(x) = |x|$ can be exactly represented by only two ReLU neurons as $|x| = \max\{x, 0\} + \max\{-x, 0\}$. Thus we show the results of width 2 on the right of Fig. 1. In contrast to the theoretical expressivity, we observe that only one out of five simulations shows the convergence. It turns out that there is a probability greater than 0.43 that the network of width 2 fails to be trained successfully (Theorem 1), see also Lu et al. (2019).

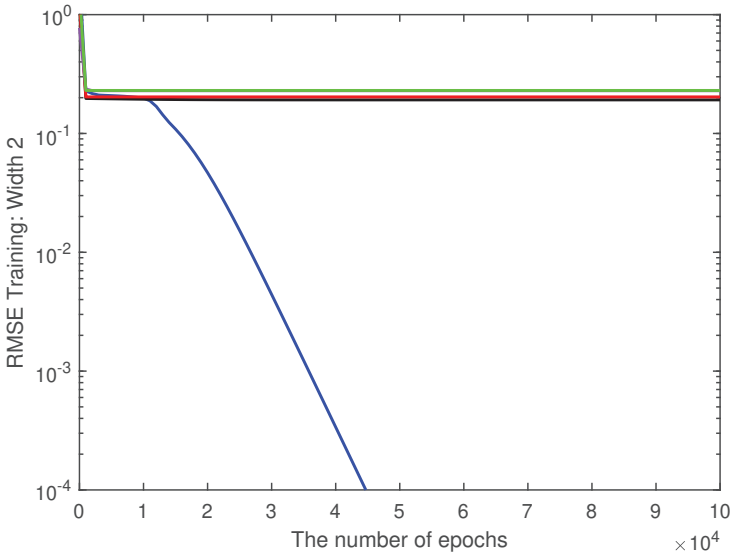
In this paper we study the trainability of ReLU networks, a *necessary* condition for successful training, and propose a data-dependent initialization for better training. Our specific contributions are summarized below:

- We classify a dead neuron into two states: tentatively dead and permanently dead. With the new classification, we introduce a notion of trainable networks (precise definition is given in Section 3). By combining it with Lemma 1, we conclude that a network being trainable is a necessary condition for successful training. That is, if an initialized ReLU network is not trainable, regardless of which gradient-based optimization method is selected, the training will not be successful.

- The probability of a randomly initialized network being trainable is referred to as *trainability* (trainable probability). We establish a general formulation of computing trainability and derive the trainabilities of ReLU networks of depths 2 and 3.



(a)



(b)

FIG. 1: The root-mean-square errors on the training data of five independent simulations with respect to the number of epochs. The standard L_2 loss is employed. (a) Width 10 and depth 2. (b) Width 2 and depth 2.

- With the computed trainability, we show that for shallow ReLU networks, overparameterization is both *a necessary and a sufficient condition* for minimizing the training loss, i.e., interpolating all training data.

- Motivated by our theoretical results, we propose a new data-dependent initialization scheme.

Taken together, our developments provide new insight into the training of ReLU neural networks that can help us design efficient network architectures and reduce the effort in optimizing the networks.

The rest of this paper is organized as follows. Upon presenting the mathematical setup in Section 2, we present the trainability of ReLU networks in Section 3. A new data-dependent initialization is introduced in Section 4. Numerical examples are provided in Section 5 before the conclusion in Section 6.

2. MATHEMATICAL SETUP

Let $\mathcal{N}^L : \mathbb{R}^{d_{\text{in}}} \mapsto \mathbb{R}^{d_{\text{out}}}$ be a feed-forward neural network with L layers and n_j neurons in the jth layer ($n_0 = d_{\text{in}} = d$, $n_L = d_{\text{out}}$). For $1 \le j \le L$, the weight matrix and the bias vector in the jth layer are denoted by $\boldsymbol{W}^j \in \mathbb{R}^{n_j \times n_{j-1}}$ and $\boldsymbol{b}^j \in \mathbb{R}^{n_j}$, respectively; n_j is called the width of the jth layer. We also denote the input by $\mathbf{x} \in \mathbb{R}^{d_{\text{in}}}$ and the output at the jth layer by $\mathcal{N}^j(\mathbf{x})$. Given an activation function ϕ which is applied element-wise, the feed-forward neural network is defined by

$$\mathcal{N}^j(\mathbf{x}) = \boldsymbol{W}^j \phi(\mathcal{N}^{j-1}(\mathbf{x})) + \boldsymbol{b}^j \in \mathbb{R}^{n_j}, \quad \text{for} \quad 2 \le j \le L,$$

and $\mathcal{N}^1(\mathbf{x}) = \boldsymbol{W}^1\mathbf{x} + \boldsymbol{b}^1$. Note that $\mathcal{N}^L(\mathbf{x})$ is called a $(L-1)$-hidden layer neural network or a L-layer neural network. Also, $\phi(\mathcal{N}_i^j(\mathbf{x}))$, $i = 1, \cdots, n_j$, is called a neuron or a unit in the jth hidden layer. We use $\boldsymbol{n} = (n_0, \cdots, n_L)$ to describe a network architecture. In this paper we refer to a two-layer network as a shallow network and a L-layer network as a deep network for $L > 2$.

Let θ be a collection of all weight matrices and bias vectors, i.e., $\theta = \{\boldsymbol{V}^j\}_{j=1}^L$ where $\boldsymbol{V}^j = [\boldsymbol{W}^j, \boldsymbol{b}^j]$. To emphasize the dependency on θ, we often denote the neural network by $\mathcal{N}^L(\mathbf{x}; \theta)$. In this paper, the ReLU is employed as an activation function, i.e.,

$$\phi(\mathbf{x}) = \text{ReLU}(\mathbf{x}) := (\max\{x_1, 0\}, \cdots, \max\{x_{d_{\text{in}}}, 0\})^T,$$

where $\mathbf{x} = (x_1, \cdots, x_{d_{\text{in}}})^T$.

In many machine learning applications, the goal is to train a neural network using a set of training data $\mathcal{T}_m$. Each datum is a pair of an input and an output, $(\mathbf{x}, \mathbf{y}) \in \mathcal{X} \times \mathcal{Y}$. Here $\mathcal{X} \subset \mathbb{R}^{d_{\text{in}}}$ is the input space and $\mathcal{Y} \subset \mathbb{R}^{d_{\text{out}}}$ is the output space. Thus we write $\mathcal{T}_m = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$. In order to measure the discrepancy between a prediction and an output, we introduce a loss metric $\ell(\cdot, \cdot) : \mathcal{Y} \times \mathcal{Y} \mapsto \mathbb{R}$ to define a loss function $\mathcal{L}$:

$$\mathcal{L}(\theta) = \frac{1}{m} \sum_{i=1}^{m} \ell(\mathcal{N}^L(\mathbf{x}_i; \theta), \mathbf{y}_i). \tag{1}$$

For example, the squared loss $\ell(\hat{\mathbf{y}}, \mathbf{y}) = \|\hat{\mathbf{y}} - \mathbf{y}\|^2$, logistic $\ell(\hat{y}, y) = \log(1 + \exp(-y\hat{y}))$, hinge, or cross-entropy are commonly employed. We then seek to find θ^*, which minimizes the loss function $\mathcal{L}$. In general, a gradient-based optimization method is employed for the training. In its very basic form, given an initial value of $\theta^{(0)}$, the parameters are updated according to

$$\theta^{(k+1)} = \theta^{(k)} - \eta_k \frac{\partial \mathcal{L}(\theta)}{\partial \theta}\bigg|_{\theta=\theta^{(k)}},$$

where η_k is the learning rate of the kth iteration. There are many stochastic variants of gradient descent (Ruder, 2016) that are popularly employed in practice. Throughout this paper, such variants are referred to as gradient-based optimization. This includes minibatch stochastic gradient descent or its variants.

2.1 Weights and Biases Initialization and Data Normalization

Gradient-based optimization is a popular choice for training a neural network. It commences with the weight and bias initialization. How to initialize the network plays a crucial role in the success of the training. Typically, the weights are randomly initialized from probability distributions. However, the biases could be set to zeros initially or could be randomly initialized.

In this paper we consider the following weights and biases initialization schemes. One is the normal initialization. That is, all weights and/or biases in the $(t+1)$th layer are independently initialized from zero-mean normal distributions:

$$\begin{aligned} &\text{(“Normal” without bias)} \quad \boldsymbol{W}_j^{t+1} \sim N\left(0, \sigma_{t+1}^2 \boldsymbol{I}_{n_t}\right), \quad \boldsymbol{b}_j^{t+1} = 0, \\ &\text{(“Normal” with bias)} \quad \boldsymbol{W}_j^{t+1} \sim N\left(0, \sigma_{t+1}^2 \boldsymbol{I}_{n_t}\right), \quad \boldsymbol{b}_j^{t+1} \sim N(0, \sigma_{b,t+1}^2), \end{aligned} \tag{2}$$

where $\boldsymbol{I}_m$ is the identity matrix of size $m \times m$. When $\sigma_{t+1}^2 = 2/n_t$ and $\boldsymbol{b}_j^{t+1} = 0$, the initialization is known as the “He initialization” (He et al., 2015). The He initialization is one of the most popular initialization methods for ReLU networks. The other initialization is from the uniform distribution on the unit hypersphere. That is, each row of either $\boldsymbol{W}^{t+1}$ or $\boldsymbol{V}^{t+1} = [\boldsymbol{W}_j^{t+1}, \boldsymbol{b}_j^{t+1}]$ is independently initialized from its corresponding unit hypersphere uniform distribution.

$$\begin{aligned} &\text{(“Unit hypersphere” without bias)} \quad \boldsymbol{W}_j^{t+1} \sim \mathrm{Unif}(\mathbb{S}^{n_t - 1}), \quad \boldsymbol{b}_j^{t+1} = 0, \\ &\text{(“Unit hypersphere” with bias)} \quad \boldsymbol{V}_j^{t+1} = [\boldsymbol{W}_j^{t+1}, \boldsymbol{b}_j^{t+1}] \sim \mathrm{Unif}(\mathbb{S}^{n_t}). \end{aligned} \tag{3}$$

Throughout this paper we assume that the training input domain is the closed ball with radius $r > 0$, i.e., $B_r(0) = \{\mathbf{x} \in \mathbb{R}^{d_{\text{in}}} | \|\mathbf{x}\|_2 \leq r\}$. In many practical applications, such as image processing or classification, there is a natural bound on the magnitude of each datum. Also, in practice, the training data is often normalized to have mean zero and/or variance 1. Given a training data set $\mathcal{T}_m = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^m$, the normalization makes $\|\mathbf{x}_i\|_2^2 \leq 1$ for all $i = 1, \cdots, m$. Thus one may assume that the training input data domain is the unit closed ball. We note that this assumption is independent of the actual data domain. This is because one can normalize the given data set since we *always* have finitely many data. Many theoretical works (Allen-Zhu et al., 2018; Du et al., 2018a; Li and Liang, 2018; Soltanolkotabi et al., 2019; Zou et al., 2018) also assume a certain data normalization. Given a training data point $\mathbf{x}_i$, let $\tilde{\mathbf{x}}_i = [\mathbf{x}_i; \alpha_i]$ for some $\alpha_i > 0$. One can then normalize $\tilde{\mathbf{x}}_i$ to have a unit norm and let $\mathbf{z}_i = \tilde{\mathbf{x}}_i / \|\tilde{\mathbf{x}}_i\|$. Here $\|\cdot\|$ is the standard Euclidean vector norm. For example, if $\|\mathbf{x}_i\| = r$ and $\alpha_i = r\sqrt{k_i - 1}$ for any $k_i > 1$, we have $\mathbf{z}_i = [\mathbf{x}_i / \sqrt{k_i r^2}; \sqrt{k_i - 1}/\sqrt{k_i}]$ whose norm is 1. In Allen-Zhu et al. (2018), k_i was chosen to be 2 for all i. To this end, $\mathbf{x}_i$ is normalized to $\mathbf{x}_i / \sqrt{k_i r^2}$. In the later sections, we will see that the choice of k_i will affect the trainability of ReLU networks.

2.2 Dying ReLU and Born Dead Probability

Dying ReLU refers to the problem when ReLU neurons become inactive and only output a constant for any input. We say that a ReLU neuron in the tth hidden layer is dead on $B_r(0)$ if it

is a constant function on $B_r(0)$. That is, there exists a constant $c \in \mathbb{R}^+ \cup \{0\}$ such that

$$\phi(\boldsymbol{w}^T \phi(\mathcal{N}^{t-1}(\mathbf{x})) + b) = c, \quad \forall \mathbf{x} \in B_r(0).$$

Also, a ReLU neuron is said to be born dead (BD) if it is dead at the initialization. In contrast, a ReLU neuron is said to be active in $B_r(0)$ if it is not a constant function on $B_r(0)$. The notion of born death was introduced in Lu et al. (2019), where a ReLU network is said to be BD if there exists a layer where all neurons are BD. We refer to the probability that a ReLU neuron is BD as the born dead probability (BDP) of a ReLU neuron.

In the first hidden layer, once a ReLU neuron is dead it cannot be revived during the training. However, a dead neuron in the tth layer where $t > 1$ could be revived by other active neurons in the same layer. In the following we provide a condition on which a dead neuron cannot be revived. The lemma is based on Lemma 10 of Lu et al. (2019).

Lemma 1. *For a shallow ReLU network ($L = 2$), none of the dead neurons can be revived through gradient-based training. For a deep ReLU network ($L > 2$), suppose the weight matrices are initialized from probability distributions, which satisfy Pr $(\boldsymbol{W}_j^t \boldsymbol{z} = 0) = 0$ for any nonzero vector $\boldsymbol{z}$. If there exists a hidden layer whose neurons are all dead, with probability 1, none of the dead neurons can be revived through gradient-based training.*

Proof. The proof can be found in Appendix A. □

3. TRAINABILITY OF RELU NETWORKS

3.1 Shallow ReLU Networks

For pedagogical reasons, we first confine ourselves to shallow (one-hidden layer) ReLU networks. For shallow ReLU networks we define the trainability as follows:

Definition 1. For a learning task that requires at least m active neurons, a shallow ReLU network of width n is said to be trainable if the number of active neurons is greater than or equal to m. If the network parameters are randomly initialized, we refer to the probability of a network being trainable at the initialization as trainability.

We note that "trainable" is a state of neural networks. The definition of "trainable" is independent of how the network was trained or initialized. As training goes on, the state may change. Different random realizations of networks may have different states. In what follows we investigate this state *from the random initialization*.

From Lemma 1, dead neurons will never be revived during the training. Thus, given a learning task which requires at least m active neurons, in order for successful training, an initialized network should have at least m active neurons in the first place. If the number of active neurons is less than m, there is no hope to train the network successfully. Therefore *a network being trainable is a necessary condition for successful training*. We note that this condition is independent of the choice of loss metric $\ell(\cdot, \cdot)$ in (1), of the number of training data, and of the choice of gradient-based optimization methods.

We now present the trainability results for shallow ReLU networks.

Theorem 1. *Given a learning task which requires a shallow ReLU network having at least m active neurons, suppose the training input domain is $B_r(0)$ and a shallow network of width $n \geq m$ is employed.*

- *If either the 'normal' (2) or the "unit hypersphere" (3) initialization without bias is used in the first hidden layer, with probability 1, the network is trainable.*
- *If either the 'normal' (2) or the "unit hypersphere" (3) initialization with bias is used in the first hidden layer, with probability,*

$$\sum_{j=m}^{n} \binom{n}{j} (1-\hat{p}_{d_{in}}(r))^j (\hat{p}_{d_{in}}(r))^{n_1-j}, \quad \hat{p}_d(r) = \frac{1}{\sqrt{\pi}} \frac{\Gamma((d+1)/2)}{\Gamma(d/2)} \int_0^{\alpha_r} (\sin u)^{d-1} du,$$

where $\alpha_r = \tan^{-1}(1/r)$*, the network is trainable. Furthermore, on average, at least*

$$n \left[1 - \sqrt{\frac{d_{in}}{2\pi}} \alpha_r (\sin \alpha_r)^{d_{in}-1} \right]$$

neurons will be active at the initialization.

Proof. The proof can be found in Appendix D. □

Theorem 1 implies that if the biases are randomly initialized, overspecification is necessary for successful training. It also shows a degree of overspecification whenever one has a specific width in mind for a learning task. If it is known (either theoretically or empirically) that a shallow network of width m can achieve a good performance, one should use a network of width $m/(1-\hat{p}_{d_{\text{in}}}(r))$ to guarantee that the initialized network has m active neurons (on average) at the initialization. For example, when $d_{\text{in}} = 1$, $r = 1/\sqrt{3}$, $m = 200$, it is suggested to work on a network of width $n = 300$ in the first place. The example (Fig. 1) given in Section 1 can be understood in this manner. By Theorem 1, with probability at least 0.43, the network of width 2 fails to be trained successfully for any learning task that requires at least two active neurons. The trainability depends only on $\hat{p}_d(r)$, which evidently shows its dependency on the maximum magnitude r of training data. The smaller r is, the larger $\hat{p}_d(r)$ becomes. This indicates that how the data are normalized also affects the trainability.

On the other hand, if the biases are initialized to zero, overparameterization or overspecification is not needed from this perspective. However, the zero-bias initialization often finds a spurious local minimum or gets stuck on a flat plateau. In Section 4, we further investigate the bias initialization.

Next we provide two concrete learning tasks that require a certain number of active neurons. For this purpose, we introduce the minimal function class.

Definition 2. Let $\mathcal{F}_n(r)$ be a class of shallow ReLU neural networks of width n defined on $B_r(0)$:

$$\mathcal{F}_n(r) = \left\{ \sum_{i=1}^{n} c_i \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i) + c_0 \,\middle|\, \forall i, \quad c_i, \quad b_i \in \mathbb{R}, \quad c_i \neq 0, \right.$$
$$\left. \boldsymbol{w}_i \in \mathbb{R}^{d_{\text{in}}}, \quad \text{and} \quad \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i) \quad \text{is active in} \quad B_r(0) \right\}.$$

Given a continuous function f and $\epsilon > 0$, a function class $\mathcal{F}_{m_\epsilon}$ is said to be the ϵ-minimal function class for f if m_ϵ is the smallest number such that $\exists g \in \mathcal{F}_{m_\epsilon}(r)$ and $|g - f| < \epsilon$ in $B_r(0)$. If $\epsilon = 0$, we say $\mathcal{F}_{m_0}(r)$ is the minimal function class for f.

We note that $\mathcal{F}_j \cap \mathcal{F}_s = \emptyset$ for $j \neq s$, and a function $f \in \mathcal{F}_j(r)$ could allow different representations in other function classes $\mathcal{F}_s(r)$ for $s > j$ in $B_r(0)$. For example, $f(x) = x$ on $B_r(0) = [-r, r]$ can be expressed as either $g_1(x) = \phi(x + r) - r \in \mathcal{F}_1(r)$, or $g_2(x) = \phi(x) - \phi(-x) \in \mathcal{F}_2(r)$. However, it cannot be represented by $\mathcal{F}_0(r)$. Thus $\mathcal{F}_1(r)$ is the minimal function class for $f(x) = x$. We remark that g_1 and g_2 are not the same function in $\mathbb{R}$; however, they are the same on $B_r(0)$. Also, note that the existence of m_ϵ in Definition 2 is guaranteed by universal function approximation theorems for shallow neural networks (Cybenko, 1989; Hornik, 1991). Hence, approximating a function whose minimal function class is $\mathcal{F}_m(r)$ is a learning task that requires at least m active neurons. Also, we say any ReLU network of width greater than m_ϵ is overspecified for approximating f within ϵ.

A network is said to be overparameterized if the number of parameters is larger than the number of training data. In this paper we consider the overparameterization, where the size of the width is greater than or equal to the number of training data. Then overparameterization can be understood under the frame of overspecification by the following lemma.

Lemma 2. *For any non-degenerate $(m+1)$ training data, there exists a shallow ReLU network of width m which interpolates all the training data. Furthermore, there exists nondegenerate $(m+1)$ training data such that any shallow ReLU network of width less than m cannot interpolate all the training data. In this sense, m is the minimal width.*

Proof. The proof can be found in Appendix B. □

Lemma 2 shows that any network of width greater than m is overspecified for interpolating $(m+1)$ training data. Thus we could regard overparameterization as a kind of overspecification. Hence, interpolating any nondegenerate $(m+1)$ training data is also a learning task that requires at least m active neurons.

With the trainability obtained in Theorem 1, we show that overparameterization is both a necessary and a sufficient condition for minimizing the loss.

Theorem 2. *For shallow ReLU networks, suppose either the normal (2) or the unit hypersphere (3) initialization with bias is employed in the first hidden layer. Also, the training input domain is $B_r(0)$. For any nondegenerate $(m+1)$ training data, which requires a network to have at least m active neurons for the interpolation, suppose m and the input dimension d_{in} satisfy*

$$1 - (1 - \delta)^{1/m} < \frac{\exp(-C_r d_{in})}{\pi d_{in}}, \quad C_r = -\log(\sin(\tan^{-1}(1/r))), \tag{4}$$

where $0 < \delta < 1$. Then overparameterization is both a necessary and a sufficient condition for interpolating all the training data with probability at least $1 - \delta$ over the random initialization by the (stochastic) gradient-descent method.

Proof. The proof can be found in Appendix C. □

We remark that Theorem 2 assumes that the biases are randomly initialized. To the best of our knowledge, all existing theoretical results also assume the random bias initialization, e.g., Du et al. (2018b), Oymak and Soltanolkotabi (2019), and Li and Liang (2018).

3.2 Trainability of Deep ReLU Networks

We now extend the notion of trainability to deep ReLU networks. Unlike dead ReLU neurons in the first hidden layer, a dead neuron in the tth hidden layer ($t > 1$) could be revived during the training if two conditions are satisfied. One is that for all layers there exists at least one active neuron. This condition is directly obtained from Lemma 1. The other is that the dead neuron should be in the condition of *tentative death*, which will be introduced shortly. We remark that these two conditions are necessary conditions for the revival of a dead neuron. We now provide a precise meaning of the tentative death as follows.

Let us consider a neuron in the tth hidden layer:

$$\phi(\boldsymbol{w}^T \mathbf{x}^{t-1} + b), \quad \mathbf{x}^{t-1} = \phi(\mathcal{N}^{t-1}(\mathbf{x})).$$

Suppose the neuron is dead. For any changes in $\mathbf{x}^{t-1}$, but not in $\boldsymbol{w}$ and b, if the neuron is still dead we say a neuron is *permanently dead*. For example, if $\boldsymbol{w}_j$, $b \leq 0$, and $t > 1$, since $\mathbf{x}^{t-1} \geq 0$, regardless of how $\mathbf{x}^{t-1}$ changes, the neuron will never be active again. Hence, in this case there is no hope that the neuron can be revived during the gradient training; otherwise we say a neuron is *tentatively dead*. Therefore any neuron is always in one of three states: active, tentatively dead, and permanently dead.

We now define the trainability for deep ReLU networks.

Definition 3. For a learning task that requires a L-layer ReLU network having at least m_t active neurons in the tth layer, a L-layer ReLU network with $\boldsymbol{n} = (n_0, n_1, \cdots, n_L)$ architecture is said to be trainable if the number of permanently dead neurons in the tth layer is less than or equal to $n_t - m_t$ for all $1 \leq t < L$. We refer to the probability of a network being trainable at the initialization as trainability.

For $L = 2$, since there is no tentatively dead neuron, Definition 1 becomes a special case of Definition 3.

We now present the trainability results for ReLU networks of depth $L = 3$ at $d_{\text{in}} = 1$. Since each layer can be initialized in different ways, we consider some combinations of them.

Theorem 3. *Suppose the training input domain is $B_r(0)$ and $d_{in} = 1$. For a learning task that requires a three-layer ReLU network having at least m_t active neurons in the tth layer, a three-layer ReLU network with $\boldsymbol{n} = (1, n_1, n_2, n_3)$ architecture is initialized as follows, (here $n_1 \geq m_1$, $n_2 \geq m_2$ and $n_3 = m_3$):*

- *Suppose the "unit hypersphere" (3) initialization without bias is used in the first hidden layer.*
  1. *If the normal (2) initialization without bias is used in the second hidden layer, with probability at least*

$$\sum_{j=m_2}^{n_2} \binom{n_2}{j} \left[\left(1 - \frac{1}{2^{n_1-1}}\right) \frac{3^j}{4^{n_2}} + \frac{1}{2^{n_1+n_2-1}} \right] + Q,$$

where

$$Q = \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \binom{n_2}{n_2-j-l, j, l} \left[\frac{(1-2^{-n_1})^{n_2-j-l}}{2^{(l+1)n_1+n_2-1}} + \left(1 - \frac{1}{2^{n_1-1}}\right) \frac{3^j (3/4 - 2^{-n_1-1})^{n_2-j-l}}{2^{(n_1+1)l+2j}} \right],$$

the network is trainable.

2. *If the normal (2) initialization with bias is used in the second hidden layers, with probability at least*

$$(1-\hat{p}_d(r))^{n_1}\left[\sum_{j=m_2}^{n_2}\binom{n_2}{j}\mathbb{E}_s\left[(1-\mathfrak{p}_2(s))^j\mathfrak{p}_2(s)^{n_2-j}\right]+Q\right],$$

where $\hat{p}_d(r)$ is defined in Theorem 1, $s\sim B(n_1,1/2)$, $\alpha_s=\tan^{-1}(s/(n_1-s))$, $g(x)=\sin(\tan^{-1}(x))$, and

$$\mathfrak{p}_2(s)=\frac{1}{2}+\left\{\int_{\pi/2}^{\pi+\alpha_s}\frac{g(r\sqrt{s}\cos(\theta))}{4\pi}d\theta+\int_{\pi+\alpha_s}^{2\pi}\frac{g(r\sqrt{n_1-s}\sin(\theta))}{4\pi}d\theta\right\},$$

$$Q=\sum_{j=1}^{m_2-1}\sum_{l=0}^{n_2-m_2}\binom{n_2}{n_2-j-l,j,l}\times\mathbb{E}_s\left[(1-\mathfrak{p}_2(s))^j(\mathfrak{p}_2(s)-2^{-n_1-1})^{n_2-j-l}(2^{-n_1-1})^l\right],$$

the network is trainable.

- *Suppose the unit hypersphere (3) initialization with bias is used in the first hidden layer.*

1. *If the normal (2) initialization without bias is used in the second hidden layer and $n_1=m_1=1$, with probability at least*

$$2^{-n_2}\sum_{j=m_2}^{n_2}\binom{n_2}{j}+\sum_{j=1}^{m_2-1}\sum_{l=0}^{n_2-m_2}\binom{n_2}{n_2-j-l,j,l}2^{-2n_2+j},$$

the network is trainable.

2. *If the normal (2) initialization with bias is used in the second hidden layers and $n_1=m_1=1$, with probability at least*

$$(1-\hat{p}_d(r))^{n_1}\left[\sum_{j=m_2}^{n_2}\binom{n_2}{j}\mathbb{E}_\omega\left[(1-\mathfrak{p}_2(\omega))^j\mathfrak{p}_2(\omega)^{n_2-j}\right]+Q\right],$$

where $\hat{p}_d(r)$ is defined in Theorem 1, $\alpha_r=\tan^{-1}(r)$, $\omega\sim\mathrm{Unif}\,(0,\pi/2+\alpha_r)$, $g(x)=\tan^{-1}(1/[\sqrt{r^2+1}\cos(x)])$, and

$$\mathfrak{p}_2(\omega)=\begin{cases}\dfrac{1}{4}+\dfrac{g(\omega-\alpha_r)}{2\pi},\\ \quad \textit{if}\quad \omega\in\left[\dfrac{\pi}{2}-\alpha_r,\dfrac{\pi}{2}+\alpha_r\right),\\ \dfrac{1}{4}+\dfrac{g(\omega-\alpha_r)+\tan^{-1}(\sqrt{r^2+1}\cos(\omega+\alpha_r))}{2\pi},\\ \quad \textit{if}\quad \omega\in\left[0,\dfrac{\pi}{2}-\alpha_r\right),\end{cases}$$

$$Q = \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \binom{n_2}{n_2-j-l, j, l} \mathbb{E}_\omega \left[(1-\mathfrak{p}_2(\omega))^j (\mathfrak{p}_2(\omega) - 2^{-2})^{n_2-j-l} (2^{-2})^l \right],$$

the network is trainable.

Proof. The proof can be found in Appendix F. □

Theorem 3 suggests us to use a ReLU network with sufficiently large width at each layer to secure a high trainability. Also, it is clear that different initialization schemes result in different trainabilities. Our proof is built on the study of the probability distribution of the number of active neurons (see Lemma 6). In Fig. E1 of Appendix E, we illustrate the active neuron distributions by three different initialization schemes.

At last, we present an upper bound of the trainability when the biases are initialized to zeros.

Corollary 1. *For a learning task that requires a L-layer ReLU network having at least m_t active neurons in the tth layer, suppose that all weights are independently initialized from the normal (2) initialization without bias, and $d_{in} = 1$. Then the trainability of a L-hidden layer ReLU network having $n \geq m_t$ neurons at each layer is bounded above by*

$$\mathfrak{a}_1^{L-1} - \frac{(1-2^{-n+1})(1-2^{-n})}{1+(n-1)2^{-n}}(-\mathfrak{a}_1^{L-1} + \mathfrak{a}_2^{L-1}),$$

where $\mathfrak{a}_1 = 1-2^{-n}$ and $\mathfrak{a}_2 = 1-2^{-n+1}-(n-1)2^{-2n}$.

Proof. The proof can be found in Appendix G. □

Further characterization will be deferred to a future study, but a general formulation is established and can be found in Lemma 8 in Appendix F.

In principle, a single active neuron in the highest layer could potentially revive tentatively dead neurons through backpropagation (gradient). However, in practice it would be better for an initialized network to have at least m_t active neurons in the tth hidden layer for both faster training and robustness. Let A be the event that a ReLU network has at least m_t active neurons in the tth hidden layer for $t = 1, \cdots, L$. The probability of A is then a naive lower bound of trainability. Hence, having a high probability of A enforces a high trainability.

Remark 1. *A trainable network itself does not guarantee successful training. However, if a network is not trainable, there is no hope for the network to be trained successfully. Thus, a network being trainable is a necessary condition for successful training, and the trainability serves as an upper bound of the training success rate. The demonstration of trainability is given in Section 5.*

Remark 2. *Definition 3 (also Definition 1) requires the number of active neurons m_t needed for a learning task. Since neural networks are universal approximators (Cybenko, 1989; Hornik, 1991), the existence of m_t's is guaranteed; however, the exact determination of m_t's is challenging for a general learning task. This is also related to the design of network architecture. In practice, m_t's could be estimated based on trial and error or the practitioner's expertise.*

4. DATA-DEPENDENT BIAS INITIALIZATION: SHALLOW ReLU NETWORKS

In this section, we investigate the bias initialization in gradient-based training. In terms of trainability for shallow ReLU networks, Theorem 1 indicates that the zero-bias initialization would be preferred over the random-bias initialization. In practice, however, the zero-bias initialization often finds a spurious local minimum or gets stuck on a flat plateau. To illustrate this difficulty, we consider a problem of approximating a sum of two sine functions $f(x) = \sin(4\pi x) + \sin(6\pi x)$ on $[-1, 1]$. For this task, we use a shallow ReLU network of width 500 with the He initialization without bias. In order to reduce extra randomness in the experiment, 100 equidistant points on $[-1, 1]$ are used as the training data set. One of the most popular gradient-based optimization methods, `Adam` (Kingma and Ba, 2015), is employed with its default parameters. We use the full-batch size and set the maximum number of epochs to 15,000. The trained network is plotted in Fig. 2. It is clear that the trained network is stuck on a local minimum. A similar behavior is repeatedly observed in all of our multiple independent simulations.

This phenomenon could be understood as follows. Since the biases are zero, all initialized neurons are clustered at the origin. Consequently, it would take a long time for a gradient update to distribute neurons over the training domain to achieve a small training loss. In the worst case, along the way of distributing neurons it will find a spurious local minimum. We refer to this problem as the *clustered neuron problem*. Indeed, this is observed in Fig. 2. The trained network well approximates the target function on a small domain containing the origin, however, it loses its accuracy on the domain far from the origin.

On the other hand, if we randomly initialize the bias, as shown in Theorem 1, overspecification is inevitable to guarantee a certain number of active neurons. In this setting, at the initialization only 375 neurons will be active among 500 neurons on average. In Fig. 2, we also show the trained result by the He initialization with bias. Since neurons are now randomly distributed over the entire domain, the trained network approximates quite well the target function.

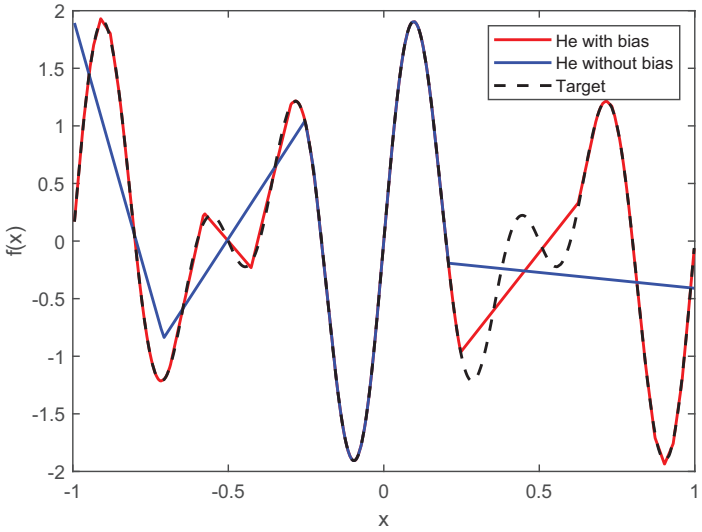


FIG. 2: The trained networks for approximating $f(x) = \sin(4\pi x) + \sin(6\pi x)$ by the He initialization without bias and with bias. A shallow ReLU network of width 500 is employed. The target function $f(x)$ is also plotted.

However, the randomness may locate some neurons in places that may lead to a spurious local minimum or a slow training. In the worst case, some neurons would never be activated. In this example the trained network by the random-bias initialization loses its accuracy at some parts of the domain, e.g., in the intervals containing ± 0.5.

In order to overcome such difficulties and accelerate the gradient-based training, we propose a new data-dependent initialization scheme. The scheme is for the overparameterized setting, where the size of width is greater than or equal to the number of training data. By adapting the trainability perspective, the method is designed to alleviate both the clustered neuron problem and the dying ReLU neuron problem at the same time. This is done by efficiently locating each neuron based on the training data.

Remark 3. *We aim to study the effect of bias initialization on gradient-based training. Interpolating all the training data results in a zero training loss. However, we do not simply attempt to interpolate the training data, which can be done by an explicit construction shown in Lemma 2. We remark that the idea of data-dependent initialization is not new; see Ioffe and Szegedy (2015), Krähenbühl et al. (2015), and Salimans and Kingma (2016). However, our method is specialized to the overparameterized setting.*

4.1 Data-Dependent Bias Initialization

Let m be the number of training data and n be the width of a shallow ReLU network. Suppose the network is overparameterized so that $n = hm$ for some positive number $h \geq 1$. We then propose to initialize the biases as follows:

$$\boldsymbol{b}_i = -\boldsymbol{w}_i^T \mathbf{x}_{j_i} + |\epsilon_i|, \quad \epsilon_i \sim N(0, \sigma_e^2),$$

where ϵ_i's are iid and $j_i - 1 = (i-1) \mod m$. We note that this mimics the explicit construction for the data interpolation in Lemma 2. By doing so, the ith neuron is initialized to be located near $\mathbf{x}_{j_i}$ as

$$\phi(\boldsymbol{w}_i^T(\mathbf{x} - \mathbf{x}_{j_i}) + |\epsilon_i|).$$

The precise value of σ_e^2 is determined as follows. Let $q(\mathbf{x})$ be the expectation of the normalized squared norm of the network, i.e., $q(\mathbf{x}) := \mathbb{E}[\|\mathcal{N}(\mathbf{x})\|_2^2]/d_{\text{out}}$, where the expectation is taken over weights and biases and $\mathcal{N}(\mathbf{x})$ is a shallow ReLU network having $\boldsymbol{n} = (d_{\text{in}}, n, d_{\text{out}})$ architecture. Given a set of training input data $\mathcal{X}_m = \{\mathbf{x}_i\}_{i=1}^m$, we define the average of $q(\mathbf{x})$ on $\mathcal{X}_m$ as

$$\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})] := \frac{1}{m}\sum_{i=1}^m q(\mathbf{x}_i).$$

We then choose our parameters to match $\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})]$ by our data-dependent initialization to the one by the standard initialization method. For example, when the normal (2) initialization without bias is used, we have

$$\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})] := \frac{n\sigma_{\text{out}}^2\sigma_{\text{in}}^2}{2m}\|\boldsymbol{X}\|_F^2, \quad \boldsymbol{X} = [\mathbf{x}_1, \cdots, \mathbf{x}_m],$$

where $\boldsymbol{W}_j^1 \sim N(0, \sigma_{\text{in}}^2 \boldsymbol{I}_{d_{\text{in}}})$ for $1 \leq j \leq n$, $\boldsymbol{W}_i^2 \sim N(0, \sigma_{\text{out}}^2 \boldsymbol{I}_n)$ for $1 \leq i \leq d_{\text{out}}$, and $\|\cdot\|_F$ is the Frobenius norm. When the "He initialization" without bias is used, i.e., $\sigma_{\text{in}}^2 = 2/d_{\text{in}}$ and $\sigma_{\text{out}}^2 = 2/n$, we have $\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})] = 2\|\boldsymbol{X}\|_F^2/(d_{\text{in}}m)$.

Theorem 4. *For a shallow network of width n, suppose $n = hm$ for some positive number $h \geq 1$, where m is the number of training data. Let $\mathcal{X}_m = \{\boldsymbol{x}_i\}_{i=1}^m$ be the set of training input data. Suppose $\boldsymbol{W}_j^1 \sim N(0, \sigma_{in}^2 \boldsymbol{I}_{d_{in}})$ for $1 \leq j \leq n$, $\boldsymbol{W}_i^2 \sim N(0, \sigma_{out}^2 \boldsymbol{I}_n)$ for $1 \leq i \leq d_{out}$, $\boldsymbol{b}^2 = 0$, and $\boldsymbol{b}^1$ is initialized by the proposed method with $\sigma_e = s\sigma_{in}$. Then*

$$\mathbb{E}_{\mathcal{X}_m}[q(\boldsymbol{x})] = \frac{h\sigma_{out}^2\sigma_{in}^2}{m\pi} \sum_{k,i=1}^{m} \left[(s^2 + \Delta_{k,i}^2)h(s/\Delta_{k,i}) + s\Delta_{k,i}\right],$$

where $h(x) = \tan^{-1}(x) + \pi/2$, and $\Delta_{k,i} = \|\boldsymbol{x}_k - \boldsymbol{x}_i\|_2$.

Proof. The proof can be found in Appendix H. □

For example, if we set σ_{in}, σ_{out}, and σ_e to be

$$\sigma_{\text{in}}^2 = \frac{2}{d_{\text{in}}}, \quad \sigma_e^2 = 0, \quad \sigma_{\text{out}}^2 = \frac{1}{h} \cdot \frac{\sum_j \|\mathbf{x}_j\|^2}{\sum_{k<i} \|\mathbf{x}_k - \mathbf{x}_i\|^2}, \tag{5}$$

$\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})]$ by the data-dependent initialization is equal to the one by the He initialization without bias.

The proposed initialization ensure that all neurons are equally distributed over the training data points. Also, it would ensure that at least one neuron will be activated at a training datum. By doing so, it would effectively avoid both the clustered neuron problem and the dying ReLU neuron problem. Furthermore, it locates all neurons in favor of the training data points with a hope that such a neuron configuration accelerates the training.

In Fig. 3, we demonstrate the performance of the proposed method in approximating the sum of two sine functions. On the left, the trained neural network is plotted, and on the right, the RMSE of the training loss are plotted with respect to the number of epochs by three different initialization methods. We remark that since the training set is deterministic and the fullbatch is used, the only randomness in the training process is from the weights and biases initialization. It can be seen that the proposed method not only results in the fastest convergence but also achieves the smallest approximation error among others. The number of dead neurons in the trained network is 127 (He with bias), 3 (He without bias), and 17 (data-dependent).

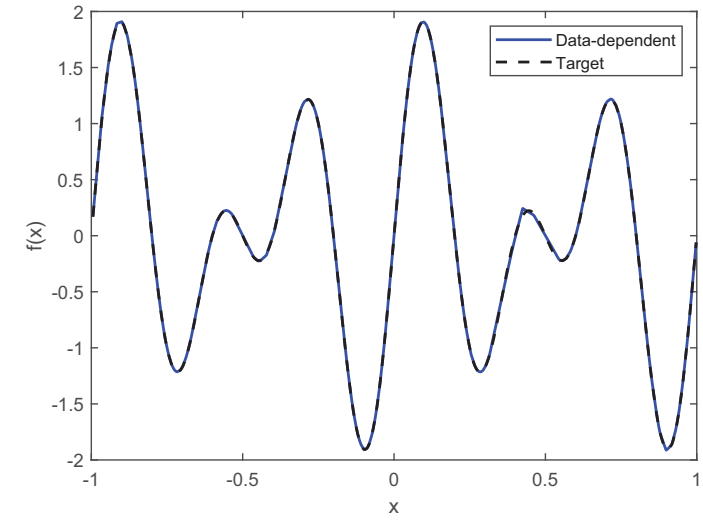
5. NUMERICAL EXAMPLES

We present numerical examples to demonstrate our theoretical findings and the effectiveness of the proposed data-dependent initialization method.

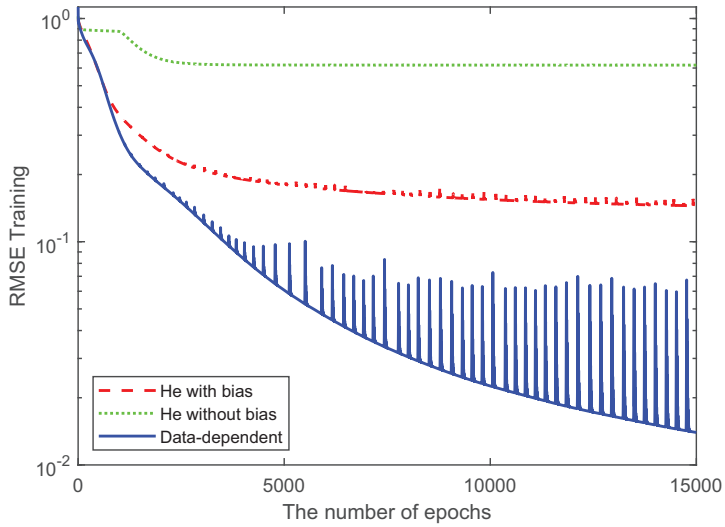
5.1 Trainability of Shallow ReLU Networks

We present two examples to demonstrate the trainability of a shallow ReLU neural network and justify our theoretical results. Here all the weights and biases are initialized according to the He initialization (2) with bias. We consider two univariate test target functions:

$$f_1(x) = |x| = \max\{x, 0\} + \max\{-x, 0\},$$
$$f_2(x) = |x| - \frac{\sqrt{3}}{\sqrt{3}-1}\max\{x-1, 0\} - \frac{\sqrt{3}}{\sqrt{3}-1}\max\{-x-1, 0\}.$$



(a)



(b)

FIG. 3: (a) The trained network for approximating $f(x) = \sin(4\pi x) + \sin(6\pi x)$ by the proposed data-dependent initialization. A shallow ReLU network of width 500 is employed. (b) The root-mean-square error of the training loss with respect to the number of epochs of `Adam` (Kingma and Ba, 2015).

We note that $\mathcal{F}_2$ is the minimal function class (see Definition 2) for $f_1(x)$, and $\mathcal{F}_4$ is the minimal function class for $f_2(x)$. That is, theoretically, f_1 and f_2 should be exactly recovered by a shallow ReLU network of width 2 and 4, respectively. For the training, we use a training set of 600 data points uniformly generated from $[-\sqrt{3}, \sqrt{3}]$ and a test set of 1,000 data points uniformly generated from $[-\sqrt{3}, \sqrt{3}]$. We employ the standard stochastic gradient descent with minibatch of size 128 and a constant learning rate of 10^{-3}. We set the maximum number of epochs to 10^6 and use the standard square loss.

In Fig. 4, we show the approximation results for approximating $f_1(x) = |x|$. On the left we plot the empirical probability of successful training with respect to the value of width. The empirical probabilities are obtained from 1,000 independent simulations, and a single simulation is regarded as a success if the test error is less than 10^{-2}. We also plot the trainability from Theorem 1. As expected, it provides an upper bound for the probability of successful training. It is clear that the more the network is overspecified, the higher trainability is obtained. Also, it can be seen that as the size of the width grows, the empirical training success rate increases. This suggests that a successful training could be achieved (with high probability) by having a very high trainability. However, since it is only a necessary condition, although an initialized network is in $\mathcal{F}_j$ for $j \geq 2$, i.e., trainable, the final trained result could be in either $\mathcal{F}_1$ or $\mathcal{F}_0$, as shown in the middle and right of Fig. 4, respectively.

Similar behavior is observed for approximating $f_2(x)$. In Fig. 5 we show the approximation results for $f_2(x)$. On the left, both the empirical probability of successful training and the trainability (Theorem 1) are plotted with respect to the size of width. Again, the trainability provides an upper bound for the probability of successful training. Also, it can be seen that the empirical training success rate increases as the width size grows. On the middle and right, we plot two of local minima which a trainable network could end up with. We remark that the choice of gradient-based optimization methods, well-tuned learning rate, and/or other tunable optimization parameters could affect the empirical training success probability. However, the maximum probability one can hope for is bounded by the trainability. In all of our simulations, we did not tune any optimization hyperparameters.

5.2 Data-Dependent Bias Initialization

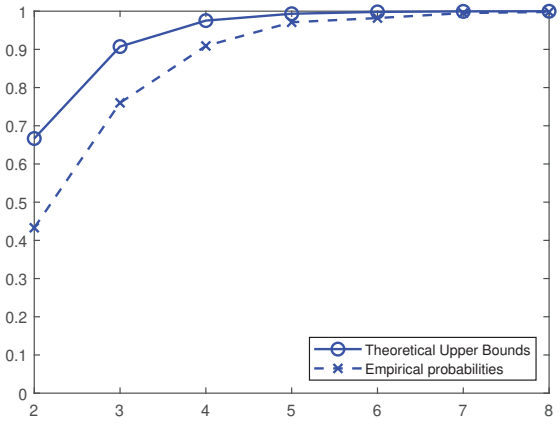
Next, we compare the training performance of three initialization methods. The first one is the He initialization (He et al., 2015) without bias. This corresponds to $\boldsymbol{W}^1 \sim N(0, 2/d_{\text{in}}), \boldsymbol{W}^2 \sim N(0, 2/n), \boldsymbol{b}^1 = 0, \boldsymbol{b}^2 = 0$. The second one is the He initialization with bias (2). This corresponds to $[\boldsymbol{W}^1, \boldsymbol{b}^1] \sim N(0, 2/(d_{\text{in}} + 1)), [\boldsymbol{W}^2, \boldsymbol{b}^2] \sim N(0, 2/(n + 1))$. Here n is the width of the first hidden layer. The last one is the proposed data-dependent initialization described in the previous section. We use the parameters from (5). All results are generated under the same conditions, except for the weights and biases initialization.

We consider the following $d_{\text{in}} = 2$ test functions on $[-1, 1]^2$:

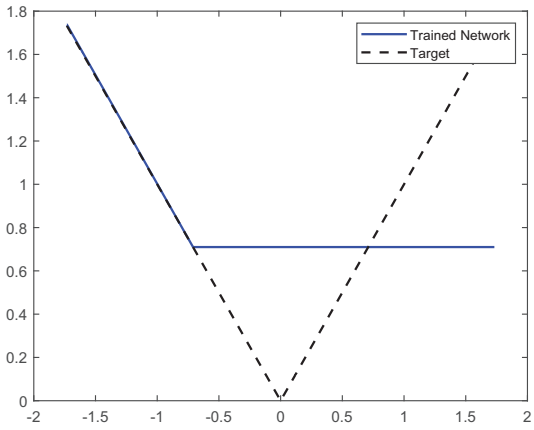
$$\begin{aligned} f_3(\mathbf{x}) &= \sin(\pi x_1)\cos(\pi x_2)e^{-x_1^2 - x_2^2}, \\ f_4(\mathbf{x}) &= \sin(\pi(x_1 - x_2))e^{x_1 + x_2}. \end{aligned} \tag{6}$$

In all tests, we employ a shallow ReLU network of width 100, and it is trained over 25 randomly uniformly drawn points from $[-1, 1]^2$. We employ the gradient-descent method with momentum with the square loss. The learning rate is a constant of 0.005, and the momentum term is 0.9.

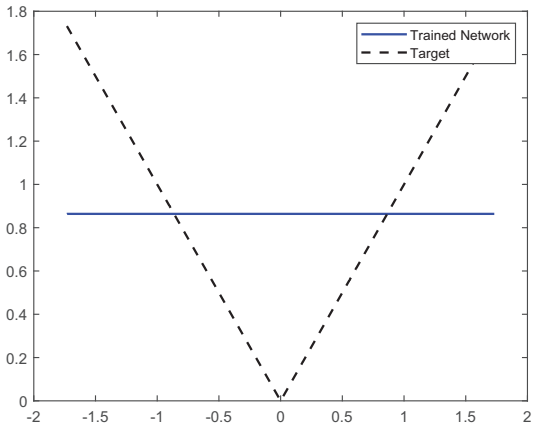
Figure 6 shows the mean of the RMSE on the training data from 10 independent simulations with respect to the number of epochs by three different initialization methods. The shaded area covers plus or minus one standard deviation from the mean. On the left and right, the results for approximating $f_3(x)$ and $f_4(x)$ are shown, respectively. We see that the data-dependent initialization not only results in the faster loss convergence but also achieves the smallest training loss. Also, the average number of dead neurons in the trained network is 11 (He with bias), 0 (He without bias), and 0 (data-dependent) for f_3, and 12 (He with bias), 0 (He without bias), and 0



(a)

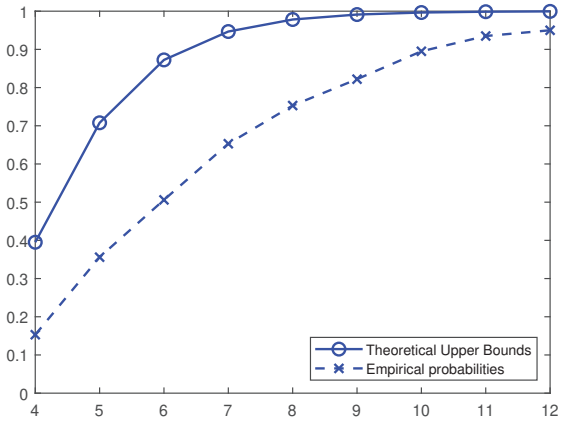


(b)

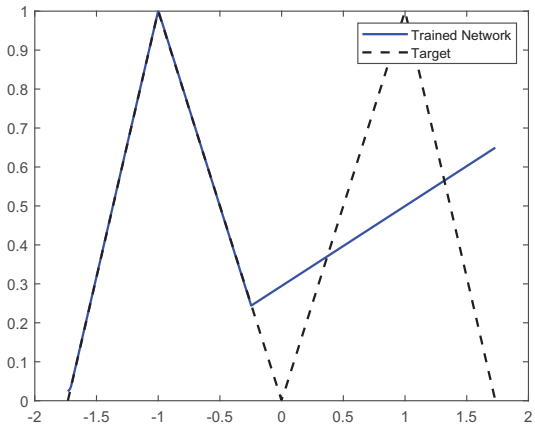


(c)

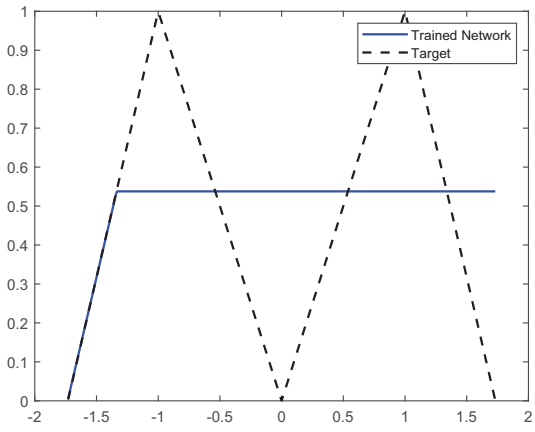
FIG. 4: (a) The empirical probability that a network approximates $f_1(x)$ successfully and the probability that a network is trainable (Theorem 1) with respect to the size of width n, and a trained network which falls in (b) $\mathcal{F}_1$ and (c) $\mathcal{F}_0$



(a)

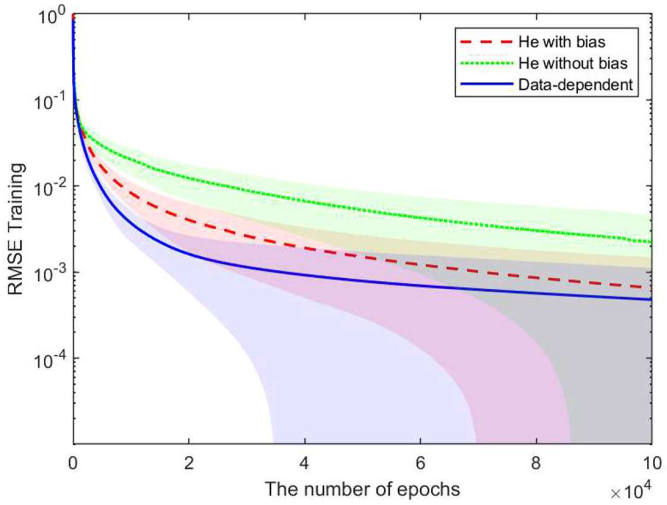


(b)

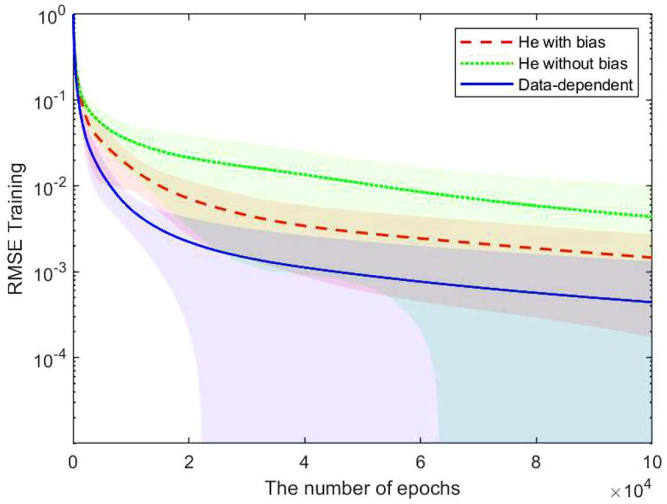


(c)

FIG. 5: (a) The empirical probability that a network approximates $f_2(x)$ successfully and the probability that a network is trainable (Theorem 1) with respect to the size of width n, and a trained network which falls in (b) $\mathcal{F}_3$ and (c) $\mathcal{F}_2$



(a)



(b)

FIG. 6: The convergence of the root-mean-square error on the training data for approximating (a) f_3 and (b) f_4 with respect to the number of epochs of the gradient descent with moment by three different initialization methods. A shallow (one-hidden layer) ReLU network of width 100 is employed. The shaded area covers plus or minus one standard deviation from the mean.

(data-dependent) for f_4. Together with the example in Section 4, all examples demonstrate the effectiveness of the proposed data-dependent initialization.

6. CONCLUSION

In this paper we establish the trainability of ReLU neural networks, a necessary condition for successful training, and propose a data-dependent initialization scheme for better training.

Upon introducing two states of dead neurons, tentatively dead and permanently dead, we define a trainable network. A network is trainable if it has sufficiently small permanently dead neurons. We show that a network being trainable is a necessary condition for the successful training. We refer to the probability of a randomly initialized network being trainable as *trainability*. The trainability serves as an upper bound of training success rates. We establish a general formulation for computing trainability and derive the trainabilities of some special cases. For shallow ReLU networks, by utilizing the computed trainability we show that overparameterization is both a neccessary and a sufficient condition for interpolating all training data, i.e., minimizing the loss.

Motivated by our theoretical results, we propose a data-dependent initialization scheme in the over-parameterized setting, where the size of width is greater than or equal to the number of training data. The proposed method is designed to avoid both the dying ReLU neuron problem and to efficiently locate all neurons at initialization for the faster training. Numerical examples are provided to demonstrate the performance of our method. We found that the data-dependent initialization method outperforms the He initialization, both with and without bias, in all of our tests.

ACKNOWLEDGMENTS

This work is supported by the DOE PhILMs project (No. de-sc0019453), the DARPA AIRA (Grant No. HR00111990025), and the AFOSR (Grant No. FA9550-17-1-0013).

REFERENCES

Allen-Zhu, Z., Li, Y., and Song, Z., A Convergence Theory for Deep Learning via Over-Parameterization, arXiv preprint, 2018. arXiv: 1811.03962

Byrd, R.H., Lu, P., Nocedal, J., and Zhu, C., A Limited Memory Algorithm for Bound Constrained Optimization, *SIAM J. Sci. Comput.*, vol. **16**, no. 5, pp. 1190–1208, 1995.

Cybenko, G., Approximation by Superpositions of a Sigmoidal Function, *Math. Control, Signals Sys.*, vol. **2**, no. 4, pp. 303–314, 1989.

Du, S.S., Lee, J.D., Li, H., Wang, L., and Zhai, X., Gradient Descent Finds Global Minima of Deep Neural Networks, arXiv preprint, 2018a. arXiv: 1811.03804

Du, S.S., Zhai, X., Poczos, B., and Singh, A., Gradient Descent Provably Optimizes Over-Parameterized Neural Networks, arXiv preprint, 2018b. arXiv: 1810.02054

Duchi, J., Hazan, E., and Singer, Y., Adaptive Subgradient Methods for Online Learning and Stochastic Optimization, *J. Machine Learning Res.*, vol. **12**, no. Jul, pp. 2121–2159, 2011.

Glorot, X. and Bengio, Y., Understanding the Difficulty of Training Deep Feedforward Neural Networks, *Int. Conf. on Artificial Intelligence and Statistics*, pp. 249–256, Sardinia, Italy, May 13–15, 2010.

He, K., Zhang, X., Ren, S., and Sun, J., Delving Deep into Rectifiers: Surpassing Human-Level Performance on Imagenet Classification, *IEEE Int. Conf. on Computer Vision*, pp. 1026–1034, Santiago, Chile, December 13–16, 2015.

Hinton, G., Overview of Mini-Batch Gradient Descent, accessed from http://www.cs.toronto.edu/∼tijmen/csc321/slides/lecture_slides_lec6.pdf, 2014.

Hinton, G., Deng, L., Yu, D., Dahl, G., Mohamed, A., Jaitly, N., Senior, A., Vanhoucke, V., Nguyen, P., and Kingsbury, B., Deep Neural Networks for Acoustic Modeling in Speech Recognition, *IEEE Signal Process. Mag.*, vol. **29**, no. 6, pp. 82–97, 2012.

Hornik, K., Approximation Capabilities of Multilayer Feedforward Networks, *Neural Networks*, vol. **4**, no. 2, pp. 251–257, 1991.

Ioffe, S. and Szegedy, C., Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift, *Proc. of the 32nd Int. Conf. on Machine Learning*, vol. **37**, pp. 448–456, 2015.

Kingma, D.P. and Ba, J., Adam: A Method for Stochastic Optimization, *Int. Conf. on Learning Representations*, San Diego, CA, USA, May 7–9, 2015.

Krähenbühl, P., Doersch, C., Donahue, J., and Darrell, T., Data-Dependent Initializations of Convolutional Neural Networks, arXiv preprint, 2015. arXiv: 1511.06856

Krizhevsky, A., Sutskever, I., and Hinton, G., Imagenet Classification with Deep Convolutional Neural Networks, *Adv. Neural Inf. Proc. Sys.*, vol. **25**, pp. 1097–1105, 2012.

LeCun, Y., Bottou, L., Orr, G.B., and Müller, K.R., Efficient Backprop, *Neural Networks: Tricks of the Trade*, Berlin–Heidelberg, Germany: Springer, pp. 9–48, 2012

Leopardi, P.C., Distributing Points on the Sphere: Partitions, Separation, Quadrature and Energy, PhD, University of New South Wales, Sydney, Australia, 2007.

Li, Y. and Liang, Y., Learning Overparameterized Neural Networks via Stochastic Gradient Descent on Structured Data, *Adv. Neural Inf. Proc. Sys.*, vol. **31**, pp. 8157–8166, 2018.

Livni, R., Shalev-Shwartz, S., and Shamir, O., On the Computational Efficiency of Training Neural Networks, *Adv. Neural Inf. Proc. Sys.*, vol. **27**, pp. 855–863, 2014.

Lu, L., Shin, Y., Su, Y., and Karniadakis, G.E., Dying ReLU and Initialization: Theory and Numerical Examples, arXiv preprint, 2019. arXiv: 1903.06733

Mishkin, D. and Matas, J., All You Need Is a Good Init, *Int. Conf. on Learning Representations*, San Juan, Puerto Rico, USA, May 2–4, 2016.

Nguyen, Q. and Hein, M., The Loss Surface of Deep and Wide Neural Networks, *Proc. of the 34nd Int. Conf. on Machine Learning*, vol. **70**, pp. 2603–2612, 2017.

Oymak, S. and Soltanolkotabi, M., Towards Moderate Overparameterization: Global Convergence Guarantees for Training Shallow Neural Networks, arXiv preprint, 2019. arXiv: 1902.04674

Reddi, S.J., Kale, S., and Kumar, S., On the Convergence of Adam and Beyond, arXiv preprint, 2019. arXiv: 1904.09237

Robbins, H. and Monro, S., A Stochastic Approximation Method, *Annals Math. Stat.*, vol. **22**, no. 3, pp. 400–407, 1951.

Ruder, S., An Overview of Gradient Descent Optimization Algorithms, arXiv preprint, 2016. arXiv: 1609.04747

Rumelhart, D.E., Hinton, G.E., and Williams, R.J., Learning Internal Representations by Error Propagation, Tech. Rep., California University San Diego, La Jolla Institute for Cognitive Science, 1985.

Safran, I. and Shamir, O., On the Quality of the Initial Basin in Overspecified Neural Networks, *Proc. of the 33rd Int. Conf. on Machine Learning*, vol. **48**, pp. 774–782, 2016.

Salimans, T. and Kingma, D.P., Weight normalization: A Simple Reparameterization to Accelerate Training of Deep Neural Networks, *Adv. Neural Inf. Proc. Sys.*, vol. **29**, pp. 901–909, 2016.

Saxe, A.M., McClelland, J.L., and Ganguli, S., Exact Solutions to the Nonlinear Dynamics of Learning in Deep Linear Neural Networks, *Int. Conf. Learning Representations*, Banff, Canada, April 14–16, 2014.

Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., Driessche, G.V.D., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., and Lanctot, M., Mastering the Game of Go with Deep Neural Networks and Tree Search, *Nature*, vol. **529**, no. 7587, p. 484, 2016.

Soltanolkotabi, M., Javanmard, A., and Lee, J.D., Theoretical Insights into the Optimization Landscape of

Over-Parameterized Shallow Neural Networks, *IEEE Transact. Inf. Theor.*, vol. **65**, no. 2, pp. 742–769, 2019.

Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al., Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation, arXiv preprint, 2016. arXiv: 1609.08144

Zou, D., Cao, Y., Zhou, D., and Gu, Q., Stochastic Gradient Descent Optimizes Over-Parameterized Deep RelU Networks, arXiv preprint, 2018. arXiv: 1811.08888

APPENDIX A. PROOF OF LEMMA 1

Proof. Suppose a ReLU neural network $\mathcal{N}(x)$ of width N is initialized to be

$$\mathcal{N}(x;\theta) = \sum_{i=1}^{n} c_i \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i) + \sum_{i=n+1}^{N} c_i \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i) + c_0,$$

where $\theta = [(c_i, \boldsymbol{w}_i, b_i)_{i=1}^N, c_0]$, and the second term on the right is a constant function on $B_r(0)$. Let $Z(x) = \sum_{i=n+1}^N c_i \phi(\boldsymbol{w}_i^T \mathbf{x} + b_i)$. Given a training data set $\mathcal{T}_m = \{(\mathbf{x}_i, y_i)\}_{i=1}^m$ where $\{\mathbf{x}_i\}_{i=1}^m \subset B_r(0)$, and a loss metric $\ell : \mathbb{R}^{d_{\text{out}}} \times \mathbb{R}^{d_{\text{out}}} \mapsto \mathbb{R}$, the loss function is $\mathcal{L}(\theta; \mathcal{T}_m) = \sum_{i=1}^m \ell[\mathcal{N}(\mathbf{x}_i;\theta), y_i]$. The gradients of the loss function $\mathcal{L}$ with respect to parameters are

$$\frac{\partial}{\partial \theta}\mathcal{L}(\theta;\mathcal{T}_m) = \sum_{(x,y)\in\mathcal{T}_m} \ell'[\mathcal{N}(x;\theta), y] \frac{\partial}{\partial \theta}\mathcal{N}(x;\theta). \tag{A.1}$$

Then for $i = 1, \cdots, N$, we have $[\partial/(\partial \boldsymbol{w}_i)]\mathcal{N}(x;\theta) = \phi'(\boldsymbol{w}_i^T \mathbf{x} + b_i)\mathbf{x}$ and $[\partial/(\partial b_i)] \times \mathcal{N}(x;\theta) = \phi'(\boldsymbol{w}_i^T \mathbf{x} + b_i)$. Since $Z(\mathbf{x})$ is a constant function on Ω, for $i = n+1, \cdots, N$, we have $\phi'(\boldsymbol{w}_i^T \mathbf{x} + b_i) = 0$ for all $\mathbf{x} \in B_r(0)$. Therefore any gradient-based optimization method does not update $(c_i, \boldsymbol{w}_i, b_i)_{i=n+1}^N$, which makes $Z(\mathbf{x})$ remain a constant function in $B_r(0)$.

It follows from Lemma 10 of Lu et al. (2019) that with probability 1, a network is initialized to be a constant function if and only if there exists a hidden layer such that all neurons are dead. Thus all dead neurons cannot be revived through gradient-based training. □

APPENDIX B. PROOF OF LEMMA 2

Proof. Given a set of nondegenerate m data, $\{\mathbf{x}_i, y_i\}_{i=1}^m$, for $d_{\text{in}} = 1$, suppose $x_1 < x_2 < \cdots < x_m$ and for $d_{\text{in}} > 1$, we choose a vector $\boldsymbol{w}$ such that $\boldsymbol{w}^T \mathbf{x}_1 < \cdots < \boldsymbol{w}^T \mathbf{x}_m$. We note that one can always find such $\boldsymbol{w}$. Let

$$S_{ij} = \{\boldsymbol{w} \in \mathbb{S}^{d_{\text{in}}-1} | \boldsymbol{w}^T(\mathbf{x}_i - \mathbf{x}_j) = 0\}, \quad i \neq j.$$

Since $\mathbf{x}_i$'s are distinct, S_{ij} is a Lebesgue measure zero set. Thus $\cup_{1\le i<j\le m} S_{ij}$ is also a measure zero set. Therefore the Lebesgue measure of $\cap_{1\le i<j\le m} S_{ij}^c$ is positive and thus it is nonempty. Then, any vector $\boldsymbol{w} \in \cap_{1\le i<j\le m} S_{ij}^c$ satisfies the condition.

We recursively define shallow ReLU networks; for $j = 0, \cdots, m$,

$$\mathcal{N}(\mathbf{x}; j) = y_1 + \sum_{i=1}^{j} c_i \phi[\boldsymbol{w}^T(\mathbf{x} - \mathbf{x}_i)], \quad c_i = \frac{y_{i+1} - \mathcal{N}(\mathbf{x}_{i+1}; i-1)}{\boldsymbol{w}^T(\mathbf{x}_{i+1} - \mathbf{x}_i)}.$$

Then it can be checked that $\mathcal{N}(\mathbf{x}_j; m-1) = y_j$ for all j. Since $\boldsymbol{w}^T(x_j - x_k) \le 0$ for all $k \ge j$, $\mathcal{N}(\mathbf{x}_j; m-1) = \mathcal{N}(\mathbf{x}_j; j-1)$. Also, since $\mathcal{N}(\mathbf{x}_j; j-1) = \mathcal{N}(\mathbf{x}_j; j-2) + c_{j-1}\boldsymbol{w}^T(\mathbf{x}_j - \mathbf{x}_{j-1})$ and $c_{j-1} = [y_j - \mathcal{N}(\mathbf{x}_j; j-2)]/[\boldsymbol{w}^T(\mathbf{x}_j - \mathbf{x}_{j-1})]$, we have $\mathcal{N}(\mathbf{x}_j; m-1) = y_j$.

Let $\{(\mathbf{x}_i, y_i)\}_{i=1}^m$ be the set of $(m+1)$ data such that $\mathbf{x}_i = \alpha_i \mathbf{x}_1$ and α_i's are distinct. Also let $\alpha_1 < \cdots < \alpha_m$ (after the reordering if necessary) and

$$\frac{y_{i+2} - y_{i+1}}{\alpha_{i+1} - \alpha_i} \neq \frac{y_{i+1} - y_i}{\alpha_{i+1} - \alpha_i} \neq \frac{y_i - y_{i-1}}{\alpha_i - \alpha_{i-1}}, \quad \forall i = 2, \cdots, m-2. \tag{B.1}$$

Suppose there exists a network $\mathcal{N}(\mathbf{x}; m-2)$ of width $(m-2)$ which interpolates all m data. We note that a shallow ReLU network is a piecewise linear function. That is, whenever a slope in a direction needs to be changed, a new neuron has to be added. Since the number of neurons is $(m-2)$, the number of slope changes is at most $(m-2)$. However, in order to interpolate the data set satisfying (B.1), the minimum number of slope changes is $m-1$. To be more precise, the network in the direction of $\mathbf{x}_1$ can be viewed as a one-dimensional network satisfying

$$\mathcal{N}(\mathbf{x}_i; m-2) = \mathcal{N}(\alpha_i \mathbf{x}_1; m-2) = y_i, \forall i.$$

Since $\mathcal{N}(s\mathbf{x}_1; m-2)$ is a network of width $(m-2)$ in one-dimensional input space (i.e., as a function of s) and it interpolates m data satisfying (B.1), there must be at least $m-1$ slope changes in the interval $[\alpha_1, \alpha_m]$. However, since $\mathcal{N}$ has only $(m-2)$ width, this is impossible. Therefore any shallow ReLU network of width less than $(m-2)$ cannot interpolate m data points which satisfy (B.1). □

APPENDIX C. PROOF OF THEOREM 2

Proof. It had been shown in several existing works (Du et al., 2018b; Li and Liang, 2018; Oymak and Soltanolkotabi, 2019) that with probability at least $1-\delta$ over the initialization, an overparameterized shallow ReLU network can interpolate all training data by the (stochastic) gradient-descent method. In other words, overparameterization is a sufficient condition for interpolating all training data with probability at least $1-\delta$.

By Lemma 2, in order to interpolate $(m+1)$ data points, a shallow ReLU network having at least width m is required. However, the probability that an initialized ReLU network of width m has m active neurons is

$$\Pr(\mathfrak{m}_1 = m) = (1 - \hat{p}_{d_{\text{in}}}(r))^m,$$

which decays exponentially in m. It follows from Lemma 3 that $\hat{p}_{d_{\text{in}}}(r) > (\sin \alpha_r)^{d_{\text{in}}}/(\pi d_{\text{in}})$ where $\alpha_r = \tan^{-1}(1/r)$. From the assumption of (4) we have

$$\Pr(\mathfrak{m}_1 = m) = (1 - \hat{p}_{d_{\text{in}}}(r))^m < \left[1 - \frac{(\sin \alpha_r)^{d_{\text{in}}}}{\pi d_{\text{in}}}\right]^m < 1 - \delta.$$

That is, the trainability is less than $1-\delta$. Therefore, overparameterization is required to guarantee, with probability at least $1-\delta$, that at least m neurons are active at the initialization. Therefore, overparameterization is a necessary condition for interpolating all training data. □

APPENDIX D. PROOF OF THEOREM 1

Proof. It follows from Lemma 4, since $\pi_0 = [0, \cdots, 0, 1]$, it suffices to compute the last row of the stochastic matrix P_1. For completeness, we set $(\mathrm{P}_1)_{i,:} = [1, 0, \cdots, 0]$ for $i = 1, \cdots, n_0$.

Suppose the He initialization without bias is used. Since $\mathbf{x} \in B_r(0)$, for any $\boldsymbol{w}$ there exists some $\mathbf{x} \in B_r(0)$ such that $\boldsymbol{w}^T\mathbf{x} > 0$. Therefore no ReLU neuron will be born dead; hence $(\mathrm{P}_1)_{n_0+1,:} = [0, \cdots, 1]$.

Suppose the He initialization with bias is used. Since each hidden neuron is independent, m_1 follows a binomial distribution $B(n_1, p)$. Here p represents the born dead probability of a single ReLU neuron in the first hidden layer. By Lemma 3, $p = \hat{p}_{n_0}(r)$, and this completes the proof. □

Lemma 3. *Suppose all training data inputs are from $B_r(0) = \{x \in \mathbb{R}^d | \|x\| \leq r\}$ and the weights and the biases are independently initialized from a zero mean normal distribution $N(0, \upsigma^2)$. Then the probability that a single ReLU neuron dies at the initialization is*

$$\hat{p}_d = \frac{1}{\sqrt{\pi}} \frac{\Gamma((d+1)/2)}{\Gamma(d/2)} \int_0^{\alpha_r} (\sin u)^{d-1} du < \frac{1}{2}, \tag{D.1}$$

where $\upalpha_r = \tan^{-1}(r^{-1})$. Furthermore,

$$\frac{1}{\pi d}(\sin \upalpha)^d \leq \hat{p}_d \leq \sqrt{\frac{d}{2\pi}} \upalpha (\sin \upalpha)^{d-1}. \tag{D.2}$$

Proof of Lemma 3. Let $\upphi(\boldsymbol{w}\mathbf{x} + b)$ be a single ReLU neuron where $\boldsymbol{w}_i, b \sim N(0, \upsigma^2)$. Note that in order for a single ReLU neuron to die in $B_r(0)$, for all $\mathbf{x} \in B_r(0)$, $\boldsymbol{w}\mathbf{x} + b < 0$. Therefore it suffices to calculate

$$\hat{p}_{d_{\text{in}}} = \Pr[\boldsymbol{w}\mathbf{x} + b < 0, \forall \mathbf{x} \in B_r(0)].$$

Let $\boldsymbol{v} = [\boldsymbol{w}, b] \in \mathbb{R}^{d_{\text{in}}+1}$. Since $\boldsymbol{w}_i$'s and b are iid normal, $\boldsymbol{s} := \boldsymbol{v}/\|\boldsymbol{v}\|$ follows the uniform distribution on the unit hypersphere $\mathbb{S}^{d_{\text{in}}}$, i.e., $\boldsymbol{s} \sim \mathcal{U}(\mathbb{S}^{d_{\text{in}}})$. Also, since $\boldsymbol{s} \overset{d}{=} -\boldsymbol{s}$, we have

$$\hat{p}_{d_{\text{in}}} = \Pr[\langle \boldsymbol{s}, [\mathbf{x}, 1] \rangle < 0, \forall \mathbf{x} \in B_r(0)] = \Pr[\langle \boldsymbol{s}, [\mathbf{x}, 1] \rangle > 0, \forall \mathbf{x} \in B_r(0)].$$

Let

$$\mathcal{A} = \left\{ \boldsymbol{s} \in \mathbb{S}^{d_{\text{in}}} \middle| \langle \boldsymbol{s}, \boldsymbol{v} \rangle > 0, \forall \boldsymbol{v} \in B_r(0) \times \{1\} \right\}. \tag{D.3}$$

Then $\hat{p}_{d_{\text{in}}} = \Pr(\mathcal{A})$. Let $\boldsymbol{s} \in \mathbb{S}^{d_{\text{in}}}$. If $\boldsymbol{s}_{d_{\text{in}}+1} \leq 0$, then $\boldsymbol{s} \notin \mathcal{A}$. This is because there exists $\boldsymbol{v} = (0, \cdots, 0, 1) \in B_r(0) \times \{1\}$ such that $\langle \boldsymbol{s}, \boldsymbol{v} \rangle \leq 0$. Suppose $\boldsymbol{s}_{d_{\text{in}}+1} > 0$ and let $r_{\boldsymbol{s}} = 1/\boldsymbol{s}_{d_{\text{in}}+1}$. Then $\tilde{\boldsymbol{s}} := (1/\boldsymbol{s}_{d_{\text{in}}+1})\boldsymbol{s} \in B_{r_{\boldsymbol{s}}}(0) \times \{1\}$.

We can express any $\mathbf{x} \in B_r(0)$ in the spherical coordinate system, i.e.,

$$\begin{aligned}
\mathbf{x}_1 &= t\cos(\uptheta_1), \\
\mathbf{x}_2 &= t\sin(\uptheta_1)\cos(\uptheta_2), \\
&\vdots \\
\mathbf{x}_{d_{\text{in}}-1} &= t\sin(\uptheta_1)\cdots\sin(\uptheta_{d_{\text{in}}-2})\cos(\uptheta_{d_{\text{in}}-1}), \\
\mathbf{x}_{d_{\text{in}}} &= t\sin(\uptheta_1)\cdots\sin(\uptheta_{d_{\text{in}}-2})\sin(\uptheta_{d_{\text{in}}-1}).
\end{aligned} \tag{D.4}$$

Since $\boldsymbol{s}$ is a uniform random variable from $\mathbb{S}^{d_{\text{in}}}$, it is coordinate-free. Thus let $\tilde{\boldsymbol{s}} = (r_{\boldsymbol{s}}, 0, \cdots, 0, 1)$ for some $0 \leq r_{\boldsymbol{s}}$ and $\boldsymbol{v} = [\mathbf{x}, 1] \in B_r(0) \times \{1\}$. Then

$$\langle \tilde{\boldsymbol{s}}, \boldsymbol{v} \rangle = 1 + r_{\boldsymbol{s}} t\cos(\uptheta_1), \quad 0 \leq t \leq r, \quad 0 \leq \uptheta_1 \leq 2\pi.$$

In order for $\boldsymbol{s} \in \mathcal{A}$, $r_{\boldsymbol{s}} < 1/r$ has to be satisfied; therefore,

$$\mathcal{A} = \left\{ \frac{\tilde{\boldsymbol{s}}}{\|\tilde{\boldsymbol{s}}\|} \in \mathbb{S}^{d_{\text{in}}} \middle| \tilde{\boldsymbol{s}} \in B_{1/r}(0) \times \{1\} \right\}.$$

Let $\text{Surf}(\mathbb{S}^d)$ be the surface area of $\mathbb{S}^d$. It is known that $\text{Surf}(\mathbb{S}^d) = 2\pi^{(d+1)/2}/\{\Gamma[(d+1)/2]\}$, where Γ is the gamma function. Then

$$\begin{aligned}\hat{p}_{d_{\text{in}}} = \Pr(\mathcal{A}) &= \frac{1}{\text{Surf}(\mathbb{S}^{d_{\text{in}}})} \int_{\mathcal{A}} d^{d_{\text{in}}+1}\mathbf{S} = \frac{1}{\text{Surf}(\mathbb{S}^{d_{\text{in}}})} \int_{\theta_{d_{\text{in}}}=0}^{2\pi} \int_{\theta_{d_{\text{in}}-1}=0}^{\pi} \cdots \int_{\theta_2=0}^{\pi} \int_{\theta_1=0}^{\alpha} d^{d_{\text{in}}+1}\mathbf{S} \\ &= \frac{\text{Surf}(\mathbb{S}^{d_{\text{in}}-1})}{\text{Surf}(\mathbb{S}^{d_{\text{in}}})} \int_0^{\alpha} \sin^{d-1}\theta d\theta,\end{aligned}$$

where $\alpha = \tan^{-1}(1/r)$ and $d^{d_{\text{in}}+1}\mathbf{S} = \sin^{d_{\text{in}}-1}\theta_1 \sin^{d_{\text{in}}-2}\theta_2 \cdots \sin\theta_{d_{\text{in}}-1} d\theta_1 d\theta_2 \cdots d\theta_{d_{\text{in}}}$. Note that $g(d) = [\text{Surf}(\mathbb{S}^{d-1})]/[\text{Surf}(\mathbb{S}^d)]$ is bounded above by $\sqrt{d/(2\pi)}$ for all $d \geq 1$ (Leopardi, 2007) and $g(d)$ is monotonically increasing. Thus we have an upper bound of $\hat{p}_d$ as $\hat{p}_d \leq \sqrt{d/(2\pi)}\alpha(\sin\alpha)^{d-1}$. For a lower bound, it can be shown that for any $\theta \in [0, \pi]$, $\int_0^{\theta} (\sin x)^{d-1} dx \geq (1/d)(\sin\theta)^d$. Thus we have

$$\hat{p}_d \geq g(d)\frac{(\sin\alpha)^d}{d} \geq g(1)\frac{(\sin\alpha)^d}{d} = \frac{(\sin\alpha)^d}{\pi d},$$

which completes the proof. □

APPENDIX E. PROBABILITY DISTRIBUTION OF THE NUMBER OF ACTIVE ReLU NEURONS

In order to calculate the trainability, we first present the results for the distribution of the number of active neurons. Understanding how many neurons will be active at the initialization is not only directly related to the trainability of a ReLU network but also suggests how much overspecification or overparameterization shall be needed for training. Given a L-layer ReLU network with $\boldsymbol{n} = (n_0, n_1, \cdots, n_L)$ architecture, let $\mathfrak{m}_t$ be the number of active neurons at the tth hidden layer and π_t be its probability distribution. Then the distribution of $\mathfrak{m}_t$ can be identified as follows.

Lemma 4. *Let $\boldsymbol{V}^t = [\boldsymbol{W}^t, \boldsymbol{b}^t]$ be the parameter (weight and bias) matrix in the tth layer. Suppose $\{\boldsymbol{V}^t\}_{t=1}^L$ is randomly independently initialized and each row of $\boldsymbol{V}^t$ is independent of any other row and follows an identical distribution. Then the probability distribution of the number of active neurons $\mathfrak{m}_j$ at the jth hidden layer can be expressed as*

$$\pi_j = \pi_0 P_1 P_2 \cdots P_j, \quad (\pi_j)_i = Pr(\mathfrak{m}_j = i), \tag{E.1}$$

where $\pi_0 = [0, \cdots, 0, 1]$, and P_t is the stochastic matrix of size $(n_{t-1}+1) \times (n_t+1)$ whose $(i+1, j+1)$ entry is $Pr(\mathfrak{m}_t = j | \mathfrak{m}_{t-1} = i)$. Furthermore, the stochastic matrix P_t is expressed as

$$(P_t)_{(i+1,j+1)} = \binom{n_t}{j} \mathbb{E}_{t-1}\left[(1 - \mathfrak{p}_t(A_{t-1}^i))^j \mathfrak{p}_t(A_{t-1}^i)^{n_t - j}\right], \tag{E.2}$$

where $\mathbb{E}_{t-1}$ is the expectation with respect to $\{\boldsymbol{W}^i, \boldsymbol{b}^i\}_{i=1}^{t-1}$, and $\mathfrak{p}_t(A_{t-1}^i)$ is the conditional born dead probability (BDP) of a neuron in the tth layer given the event where exactly i neurons are active in the $(t-1)$th layer.

Proof of Lemma 4. By the law of total probability, it readily follows that for $j = 0, \cdots, n_t$,

$$\Pr(\mathfrak{m}_t = j) = \sum_{k=0}^{n_{t-1}} \Pr(\mathfrak{m}_t = j | \mathfrak{m}_{t-1} = k) \Pr(\mathfrak{m}_{t-1} = k),$$

which gives $\pi_t = \pi_{t-1} \mathrm{P}_t$. By recursively applying it, we obtain $\pi_t = \pi_0 \mathrm{P}_1 \cdots \mathrm{P}_t$.

For each t and i, let A_{t-1}^i be the event where exactly i neurons are active in the $(t-1)$th layer and D_t^k be the event where the kth neuron in the tth layer is dead. Since each row of $\boldsymbol{V}^t$ is iid and $\{\boldsymbol{V}^t\}_{t=1}^L$ is independent, $\Pr(D_t^k | A_{t-1}^i) = \Pr(D_t^j | A_{t-1}^i)$ for any k, j. We denote the conditional BDP of a neuron in the tth layer given A_{t-1}^i as $\mathfrak{p}_t(A_{t-1}^i)$. From the independent row assumption, the stochastic matrix P_t can then be expressed as

$$\Pr(\mathfrak{m}_t = j | \mathfrak{m}_{t-1} = i) = (\mathrm{P}_t)_{(i+1,j+1)} = \binom{n_t}{j} \mathbb{E}_{t-1} \left\{ [1 - \mathfrak{p}_t(A_{t-1}^i)]^j \mathfrak{p}_t(A_{t-1}^i)^{n_t - j} \right\},$$

where $\mathbb{E}_{t-1}$ is the expectation with respect to $\{\boldsymbol{W}^i, \boldsymbol{b}^i\}_{i=1}^{t-1}$. □

Lemma E.2 indicates that $\mathfrak{p}_t(A_{t-1}^i)$ is a fundamental quantity for the complete understanding of π_j. As a first step toward understanding π_j, we calculate the exact probability distribution π_1 of the number of active neurons in the first hidden layer.

Lemma 5. *Given a ReLU network having $\boldsymbol{n} = (n_0, n_1, \cdots, n_L)$ architecture, suppose the training input domain is $B_r(0)$. If either the normal (2) or the unit hypersphere (3) initialization without bias is used in the first hidden layer, we have*

$$(\pi_1)_j = \Pr(\mathfrak{m}_1 = j) = \delta_{j,n_1}.$$

If either the normal (2) or the unit hypersphere (3) with bias is used in the first hidden layer, $\mathfrak{m}_1$ follows a binomial distribution with parameters n_1 and $1 - \hat{p}_{n_0}(r)$, where

$$\hat{p}_d(r) = \frac{1}{\sqrt{\pi}} \frac{\Gamma((d+1)/2)}{\Gamma(d/2)} \int_0^{\alpha_r} (\sin\theta)^{d-1} d\theta, \quad \alpha_r = \tan^{-1}(r^{-1}), \tag{E.3}$$

and $\Gamma(x)$ is the Gamma function.

Proof. The proof readily follows from Lemma 3. □

We now calculate π_2 for a ReLU network at $d_{\text{in}} = 1$. Since the bias in each layer can be initialized in different ways, we consider some combinations of them.

Lemma 6. *Given a ReLU network having $\boldsymbol{n} = (1, n_1, n_2, \cdots, n_L)$ architecture, suppose the training input domain is $B_r(0)$.*

- *Suppose the unit hypersphere (3) initialization without bias is used in the first hidden layer.*
  1. *If the normal (2) initialization without bias is used in the second hidden layer, the stochastic matrix P_2 is $(P_2)_{i,:} = [1, 0, \cdots, 0]$ for $1 \le i \le n_1$ and*

$$(P_2)_{n_1+1,j+1} = \binom{n_2}{j} \left[\left(1 - \frac{1}{2^{n_1-1}}\right) \frac{3^j}{4^{n_2}} + \frac{1}{2^{n_1+n_2-1}} \right], \quad 0 \le j \le n_2.$$

2. *If the normal (2) initialization with bias is used in the second hidden layers, the stochastic matrix P_2 is $(P_2)_{i,:} = [1, 0, \cdots, 0]$ for $1 \le i \le n_1$ and*

$$(P_2)_{n_1+1,j+1} = \binom{n_2}{j} \mathbb{E}_s \left[(1 - \mathfrak{p}_2(s))^j \mathfrak{p}_2(s)^{n_2-j} \right], \quad 0 \le j \le n_2,$$

where $s \sim B(n_1, 1/2)$, $\alpha_s = \tan^{-1}(\frac{s}{n_1 - s})$, $g(x) = \sin(\tan^{-1}(x))$, and

$$\mathfrak{p}_2(s) = \frac{1}{2} + \left[\int_{\pi/2}^{\pi+\alpha_s} \frac{g(r\sqrt{s}\cos(\theta))}{4\pi} d\theta + \int_{\pi+\alpha_s}^{2\pi} \frac{g(r\sqrt{n_1 - s}\sin(\theta))}{4\pi} d\theta \right].$$

- *Suppose the unit hypersphere (3) initialization with bias is used in the first hidden layer.*
  1. *If the normal (2) initialization without bias is used in the second hidden layer and $n_1 = 1$, the stochastic matrix P_2 is $(P_2)_{1,:} = [1, 0, \cdots, 0]$ and*

$$(P_2)_{2,:} = \textit{Binomial}(n_2, 1/2).$$

  2. *If the normal (2) initialization with bias is used in the second hidden layers and $n_1 = 1$, the stochastic matrix P_2 is $(P_2)_{1,:} = [1, 0, \cdots, 0]$ and*

$$(P_2)_{2,j+1} = \binom{n_2}{j} \mathbb{E}_\omega \left[(1 - \mathfrak{p}_2(\omega))^j \mathfrak{p}_2(\omega)^{n_2-j} \right], \quad 0 \le j \le n_2,$$

where $\alpha_r = \tan^{-1}(r)$, $\omega \sim \mathrm{Unif}\,(0, \pi/2 + \alpha_r)$, $g(x) = \tan^{-1}(1/\sqrt{r^2+1}\cos(x))$, and

$$\mathfrak{p}_2(\omega) = \begin{cases} \dfrac{1}{4} + \dfrac{g(\omega - \alpha_r)}{2\pi}, \\ \quad \textit{if} \quad \omega \in \left[\dfrac{\pi}{2} - \alpha_r, \dfrac{\pi}{2} + \alpha_r \right), \\ \dfrac{1}{4} + \dfrac{g(\omega - \alpha_r) + \tan^{-1}(\sqrt{r^2+1}\cos(\omega + \alpha_r))}{2\pi}, \\ \quad \textit{if} \quad \omega \in \left[0, \dfrac{\pi}{2} - \alpha_r \right). \end{cases}$$

Then $\pi_2 = \pi_1 P_2$, where π_1 is defined in Lemma 5.

Proof of Lemma 6. Since π_1 is completely characterized in Lemma 5, it suffices to calculate the stochastic matrix P_2, as $\pi_2 = \pi_1 \mathrm{P}_2$. From Eq. (E.2), it suffices to calculate the BDP $\mathfrak{p}_2(A_1^i)$ of a ReLU neuron at the second layer given A_1^i.

We note that if $\boldsymbol{z} = [x, 1]$ where $x \in B_r(0) = [-r, r]$ and $\boldsymbol{v} = [w, b] \sim \mathcal{U}(\mathbb{S}^1)$, then

$$\begin{aligned} P(\phi(\boldsymbol{v}^T \boldsymbol{z}) = \boldsymbol{v}^T \boldsymbol{z}, \forall \boldsymbol{z} \in B_r(0) \times \{1\}) &= \hat{p}_1(r) = \frac{\tan^{-1}(1/r)}{\pi}, \\ P(\phi(\boldsymbol{v}^T \boldsymbol{z}) = \boldsymbol{v}^T \boldsymbol{z} \mathbb{I}_{x \in [a,b]}, \forall \boldsymbol{z} \in B_r(0) \times \{1\}) &= \frac{1}{4} + \frac{\tan^{-1}(1/b) + \tan^{-1}(a)}{2\pi}. \end{aligned} \tag{E.4}$$

First, let us consider the case where the unit hypersphere initialization without bias is used for the first hidden layer. Note that since $x \in [-r, r]$, i.e., $d_{\text{in}} = 1$, we have $A_1^j = \emptyset$ for $0 \le j < n_1$

and $A_1^{n_1} = \{1, -1\}^{n_1}$. Also, note that if w_j's are iid normal, $\sum_{j=1}^{s} w_j \overset{d}{=} \sqrt{s}w$, where $w \overset{d}{=} w_1$. For fixed $\omega \in A_1^{n_1}$, a single neuron in the second layer is

$$\phi\left[\sum_{j=1}^{s} \boldsymbol{w}_{1,j}^2 \phi(x) + \sum_{j=s+1}^{n_1} \boldsymbol{w}_{1,j}^2 \phi(-x) + b\right] \overset{d}{=} \phi\Big[\sqrt{s}w_1\phi(x) + \sqrt{n_1 - s}w_2\phi(-x) + b\Big], \tag{E.5}$$

where s is the number of 1's in ω. If the normal initialization without bias is used for the second hidden layer, we have

$$\mathfrak{p}_2(\omega) = \begin{cases} \dfrac{1}{2}, & \text{if} \quad \omega = \pm[1, \cdots, 1]^T, \\ \dfrac{1}{4}, & \text{otherwise.} \end{cases}$$

Also, $s \sim \text{Binomial}(n_1, 1/2)$. Thus, for $j = 0, \cdots, n_2$,

$$\begin{aligned} \Pr(\mathrm{m}_2 = j | \mathrm{m}_1 = n_1) &= \binom{n_2}{j} \mathbb{E}_1\left[(1 - \mathfrak{p}_2(A_1^{n_1}))^j (\mathfrak{p}_2(A_1^{n_1}))^{n_2 - j}\right] \\ &= \binom{n_2}{j} \mathbb{E}_s\left[(1 - \mathfrak{p}_2(A_1^{n_1}))^j (\mathfrak{p}_2(A_1^{n_1}))^{n_2 - j}\right] \\ &= \binom{n_2}{j}\left[\frac{1}{2^{n_1-1}}\frac{1}{2^{n_2}} + \left(1 - \frac{1}{2^{n_1-1}}\right)\frac{3^j}{4^{n_2}}\right]. \end{aligned}$$

Suppose the normal initialization with bias is used for the second hidden layer. It follows from (E.5) that

$$\mathfrak{p}_2(\omega) = \Pr(w_1\sqrt{s}\phi(x) + w_2\sqrt{n_1 - s}\phi(-x) + b < 0, \forall x \in [-r, r] | \boldsymbol{W}^1 \text{ has } s \text{ 1's}).$$

Let $\boldsymbol{z} = [\sqrt{s}\phi(x), \sqrt{n_1 - s}\phi(-x), 1]$ and $\boldsymbol{v} = (w_1, w_2, b)$. Without loss of generality, we normalize $\boldsymbol{v}$. Then $\boldsymbol{v} \sim \mathbb{S}^2$, and we write it as

$$\boldsymbol{v} = (\cos\theta \sin\alpha, \sin\theta \sin\alpha, \cos\alpha),$$

where $\theta \in [0, 2\pi]$ and $\alpha \in [0, \pi]$. Since $\boldsymbol{v} \overset{d}{=} -\boldsymbol{v}$, it suffices to compute

$$\Pr(\boldsymbol{v}^T\boldsymbol{z} > 0, \forall \boldsymbol{z} | \boldsymbol{W}^1 \text{ has } s \text{ 1's}).$$

Also, note that

$$\begin{aligned} \boldsymbol{v}^T\boldsymbol{z} &= \begin{cases} \sqrt{s}\phi(x)\cos\theta\sin\alpha + \cos\alpha, & \text{if} \quad x > 0, \\ \sqrt{n_1 - s}\phi(-x)\sin\theta\sin\alpha + \cos\alpha, & \text{if} \quad x < 0, \end{cases} \\ &= \begin{cases} \sqrt{1 + sx^2\cos^2\theta}\cos(\alpha - \beta), & \text{if} \quad x > 0, \\ \sqrt{1 + (n_1 - s)x^2\sin^2\theta}\cos(\alpha - \beta), & \text{if} \quad x < 0, \end{cases} \end{aligned}$$

where $\tan\beta = \sqrt{s}x\cos\theta$ if $x > 0$ and $\tan\beta = \sqrt{n_1 - s}\sin\theta$ if $x < 0$. Given $\boldsymbol{W}^1$ which has s 1's, the regime in $\mathbb{S}^2$, where $\boldsymbol{v}^T\boldsymbol{z} > 0$ for all $\boldsymbol{z}$, is

$$\begin{aligned}
&\text{for } \omega \in \left[0, \frac{\pi}{2}\right], \alpha \in \left[0, \frac{\pi}{2}\right], \\
&\text{for } \omega \in \left[\frac{\pi}{2}, \pi + \omega^*\right], \alpha \in \left[0, \tan^{-1}(\sqrt{s}r\cos\theta) + \frac{\pi}{2}\right], \\
&\text{for } \omega \in [\pi + \omega^*, 2\pi], \alpha \in \left[0, \tan^{-1}(\sqrt{n_1 - s}r\sin\theta) + \frac{\pi}{2}\right],
\end{aligned} \tag{E.6}$$

where $\tan\omega^* = s/(n_1 - s)$. By uniformly integrating the above domain in $\mathbb{S}^2$, we have

$$\mathfrak{p}_2(s) = \frac{1}{2} + \left[\int_{\pi/2}^{\pi+\alpha_s} \frac{g(r\sqrt{s}\cos(\theta))}{4\pi}d\theta + \int_{\pi+\alpha_s}^{2\pi} \frac{g(r\sqrt{n_1 - s}\sin(\theta))}{4\pi}d\theta\right],$$

where $g(x) = \sin[\tan^{-1}(x)]$. Thus we obtain

$$(\mathrm{P}_2)_{n_1+1,j+1} = \binom{n_2}{j}\mathbb{E}_s\left[(1 - \mathfrak{p}_2(s))^j\mathfrak{p}_2(s)^{n_2-j}\right], \quad 0 \le j \le n_2.$$

Secondly, let us consider the case where the unit hypersphere initialization with bias is used for the first hidden layer and $n_1 = 1$. Since $[w_1, b_1] \sim \mathbb{S}^1$, we write it as $(\sin\omega, \cos\omega)$ for $\omega \in [-\pi, \pi]$. Since $x \in [-r, r]$, we have

$$\begin{aligned}
&A_1^0 = \{\omega \in [-\pi, \pi] | \phi(\sin\omega x + \cos\omega) = 0, \forall x \in [-r, r]\} = [-\pi + \alpha_r, \pi - \alpha_r], \\
&A_1^1 = (A_1^0)^c = (-\pi + \alpha_r, \pi - \alpha_r),
\end{aligned} \tag{E.7}$$

where $\alpha_r = \tan^{-1}(r)$. If the normal initialization without bias is used for the second hidden layer, since a single neuron in the second layer is $\phi[w^2\phi(w^1x + b^1)]$, for given A_1^1, we have $\mathfrak{p}_2(A_1^1) = 1/2$. Thus

$$\Pr(\mathrm{m}_2 = j|\mathrm{m}_1 = 1) = \binom{n_2}{j}(1/2)^j(1/2)^{n_2-j}, \quad j = 0, \cdots, n_2.$$

If the normal initialization with bias is used for the second hidden layer, it follows from Lemma 7 that for $\omega \in A_1^1$,

$$\mathfrak{p}_2(\omega) = \begin{cases} \dfrac{1}{4} + \dfrac{g(|\omega| - \alpha_r)}{2\pi}, \\ \quad \text{if} \quad |\omega| \in \left[\dfrac{\pi}{2} - \alpha_r, \dfrac{\pi}{2} + \alpha_r\right), \\ \dfrac{1}{4} + \dfrac{g(|\omega| - \alpha_r) + \tan^{-1}\left(\sqrt{r^2+1}\cos(|\omega| + \alpha_r)\right)}{2\pi}, \\ \quad \text{if} \quad |\omega| \in \left[0, \dfrac{\pi}{2} - \alpha_r\right), \end{cases}$$

where $g(x) = \tan^{-1}[1/(\sqrt{r^2+1}\cos(x))]$.

Thus we have

$$\Pr(\mathrm{m}_2 = j|\mathrm{m}_1 = 1) = \binom{n_2}{j}\mathbb{E}_\omega\left[(1 - \mathfrak{p}_2(\omega))^j(\mathfrak{p}_2(\omega))^{n_2-j}\right], \quad j = 0, \cdots, n_2,$$

where $\omega \sim \mathrm{Unif}(A_1^1)$. Then the proof is completed once we have the following lemma.

Lemma 7. *Given a ReLU network having $\boldsymbol{n} = (1, 1, n_2, \cdots, n_L)$, suppose $(w^1, b^1), (w^2, b^2) \sim \mathbb{S}^1$. Given $\{w^1, b^1\}$, let ω be the angle of (w^1, b^1) in $\mathbb{R}^2$. Then the BDP for a ReLU neuron at the second hidden layer is*

$$\mathfrak{p}_2(\omega) = \begin{cases} 1, \quad \textit{if} \quad |\omega| \in [\pi/2 + \alpha_r, \pi], \\ \dfrac{1}{4} + \dfrac{g(|\omega| - \alpha_r)}{2\pi}, \quad \textit{if} \quad |\omega| \in [\pi/2 - \alpha_r, \pi/2 + \alpha_r), \\ \dfrac{1}{4} + \dfrac{g(|\omega| - \alpha_r) + \tan^{-1}(\sqrt{r^2+1}\cos(|\omega| + \alpha_r))}{2\pi}, \\ \quad \textit{if} \quad |\omega| \in [0, \pi/2 - \alpha_r)\,, \end{cases}$$

where $g(x) = \tan^{-1}\{1/[\sqrt{r^2+1}\cos(x)]\}$ and $\alpha_r = \tan^{-1}(r)$.

Proof of Lemma 7. For a fixed $\boldsymbol{v} = [w, b]$ and $\boldsymbol{z} = [x, 1]$, we can write

$$\phi(\boldsymbol{v}^T\boldsymbol{z}) = \|\boldsymbol{z}\|\phi(\boldsymbol{v}^T\boldsymbol{z}/\|\boldsymbol{z}\|) = \|z\|\phi(\cos(\omega - \theta(x))), \quad \theta(x) = \tan^{-1}(x).$$

Since $\boldsymbol{v}$ is uniformly drawn from $\mathbb{S}^1$, it is equivalent to draw $\omega \sim \mathcal{U}(-\pi, \pi)$. Let $0 < \theta_{\max} = \tan^{-1}(r) < \pi/2$. Then

$$\phi(\boldsymbol{v}^T\boldsymbol{z}) = \begin{cases} \boldsymbol{v}^T\boldsymbol{z}, & \forall\theta(x), \text{ if } \omega \in \left(-\dfrac{\pi}{2} + \theta_{\max}, \dfrac{\pi}{2} - \theta_{\max}\right), \\ 0, & \forall\theta(x), \text{ if } \omega \in \left[-\pi, \dfrac{\pi}{2} - \theta_{\max}\right] \cup \left[\dfrac{\pi}{2} + \theta_{\max}, \pi\right], \end{cases}$$

and if

$$\omega \in \left(-\frac{\pi}{2} - \theta_{\max}, -\frac{\pi}{2} + \theta_{\max}\right] \cup \left[\frac{\pi}{2} - \theta_{\max}, \frac{\pi}{2} + \theta_{\max}\right),$$

we have $\phi(\boldsymbol{v}^T\boldsymbol{z}) = \boldsymbol{v}^T\boldsymbol{z}\mathbb{I}_{A(\omega)}(\theta(x))$, where $A(\omega) = \{\theta \in [-\theta_{\max}, \theta_{\max}] || \theta(x) - \omega| \leq \pi/2\}$. Due to symmetry, let us assume that $\omega \sim \mathcal{U}(0, \pi)$. Then it can be checked that $A_1^0 = [\pi/2 + \theta_{\max}, \pi]$ and $A_1^1 = [0, \pi/2 + \theta_{\max})$. Furthermore,

$$\max_{\boldsymbol{z}} \phi(\boldsymbol{v}^T\boldsymbol{z}) = \begin{cases} \sqrt{r^2+1}\cos(\omega - \theta_{\max}), & \text{if} \quad \omega \in \left(0, \dfrac{\pi}{2} + \theta_{\max}\right), \\ 0, & \text{if} \quad \omega \in \left[\dfrac{\pi}{2} + \theta_{\max}, \pi\right], \end{cases}$$

and

$$\min_{\boldsymbol{z}} \phi(\boldsymbol{v}^T\boldsymbol{z}) = \begin{cases} \sqrt{r^2+1}\cos(\omega + \theta_{\max}), & \text{if} \quad \omega \in \left(0, \dfrac{\pi}{2} - \theta_{\max}\right), \\ 0, & \text{if} \quad \omega \in \left[\dfrac{\pi}{2} - \theta_{\max}, \pi\right]. \end{cases}$$

For a fixed ω, let $\mathfrak{p}_2(\omega)$ be the probability that a single neuron at the second layer is born dead, i.e.,

$$\mathfrak{p}_2(\omega) = \Pr[w^2\phi(w^1x + b^1) + b^2 < 0, \quad \forall x \in B_r(0)|w^1, b^1].$$

Also, since $(w^2, b^2) \overset{d}{=} (-w^2, -b^2)$, we have

$$\mathfrak{p}_2(\omega) = \Pr[w^2\phi(w^1x + b^1) + b^2 > 0, \quad \forall x \in B_r(0)|w^1, b^1].$$

It follows from (E.4) that

$$\mathfrak{p}_2(\omega) = \frac{1}{4} + \frac{\tan^{-1}[1/\max_{\boldsymbol{z}} \phi(\boldsymbol{v}^T\boldsymbol{z})] + \tan^{-1}[\min_{\boldsymbol{z}} \phi(\boldsymbol{v}^T\boldsymbol{z})]}{2\pi}.$$

Thus we obtain

$$\mathfrak{p}_2(\omega) = \begin{cases} 1, \quad \text{if} \quad \omega \in \left[\frac{\pi}{2} + \theta_{\max}, \pi\right], \\ \frac{1}{4} + \frac{g(\omega - \theta_{\max})}{2\pi}, \quad \text{if} \quad \omega \in \left[\frac{\pi}{2} - \theta_{\max}, \frac{\pi}{2} + \theta_{\max}\right), \\ \frac{1}{4} + \frac{g(\omega - \theta_{\max}) + \tan^{-1}\left(\sqrt{r^2+1}\cos(\omega + \theta_{\max})\right)}{2\pi}, \\ \quad \text{if} \quad \omega \in \left[0, \frac{\pi}{2} - \theta_{\max}\right), \end{cases}$$

where $g(x) = \tan^{-1}[1/(\sqrt{r^2+1}\cos(x))]$, and this completes the proof. □

□

Lemmas 5 and 6 indicate that the bias initialization could drastically change the active neuron distributions π_j. Since $\pi_j = \pi_1 \mathrm{P}_2 \cdots \mathrm{P}_j = \pi_2 \mathrm{P}_3 \cdots \mathrm{P}_j$, the behaviors of π_1 and π_2 affect the higher layer's distributions π_j. In Fig. E1, we consider a ReLU network with $\boldsymbol{n} = (1, 6, 4, 2, n_4, \cdots, n_L)$ architecture and plot the empirical distributions π_j, $j = 1, 2, 3$, from 10^6 independent simulations at $r = 1$. On the left and the middle, the unit hypersphere (3) initializations without and with bias are employed, respectively, in all layers.

On the right, the unit hypersphere initialization without bias is employed in the first hidden layer, and the normal (2) initialization with bias is employed in all other layers. The theoretically derived distributions, π_1, π_2, are also plotted as references. We see that all empirical results are well matched with our theoretical derivations. When the first hidden layer is initialized with bias, with probability 0.8, at least one neuron in the first hidden layer will be dead. On the other hand, if the first hidden layer is initialized without bias, with probability 1, no neuron will be dead. It is clear that the distributions obtained by three initialization schemes show different behavior.

APPENDIX F. A GENERAL FORMULATION FOR COMPUTING TRAINABILITY

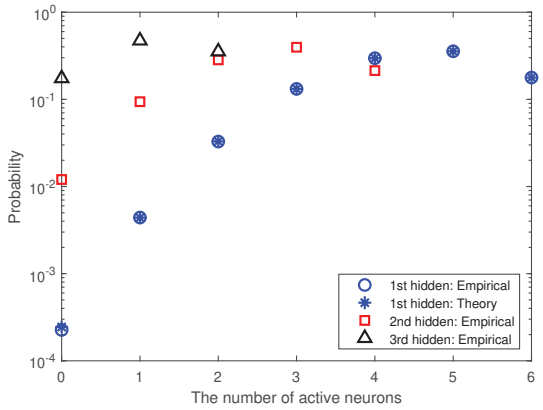
We present a general formulation for computing trainability. Our formulation requires a complete understanding of two types of inhomogeneous stochastic matrices.

Let $\mathfrak{d}_t^b$ be the number of permanently dead neurons at the tth hidden layer. Given $\{n_t, m_t\}_{t=1}^{L-1}$, let $s_t = (n_t - m_t + 1)(m_t - 1)$ for $t > 1$ and $s_1 = n_1 - m_1 + 1$. For convenience, let $\mathcal{T}_{t-1} := [\hat{n}_1] \times [\hat{n}_2] \times [\hat{m}_2] \cdots \times [\hat{n}_{t-1}] \times [\hat{m}_{t-1}]$, where $[\hat{n}_t] = \{0, \cdots, n_t - m_t\}$ and $[\hat{m}_t] = \{1, \cdots, m_t - 1\}$. Let $\hat{T}_t := [\hat{n}_t] \times [\hat{m}_t]$. Let

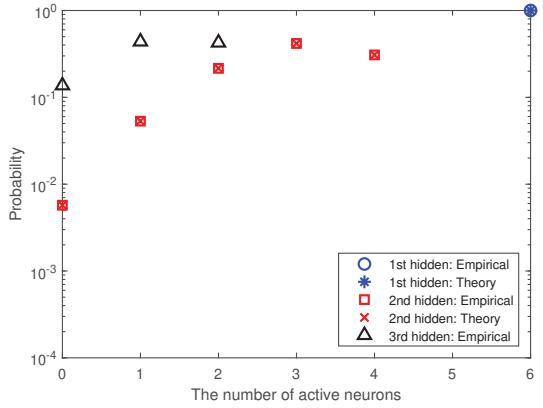
$$\mathrm{k}_{t-1}^l = (k_1^l, k_2^l, k_{2,b}^l, \cdots, k_{t-1}^l, k_{t-1,b}^l)$$

be the lth multi-index of $\mathcal{T}_{t-1}$ (assuming a certain ordering). For $1 < t < L-1$, let $\hat{P}_t$ be a matrix of size $\prod_{j=1}^{t-1} s_j \times \prod_{j=1}^{t} s_j$ defined as follows. For $l = 1, \ldots, \prod_{j=1}^{t-1} s_j$ and $r = 1, \ldots, \prod_{j=1}^{t} s_j$,

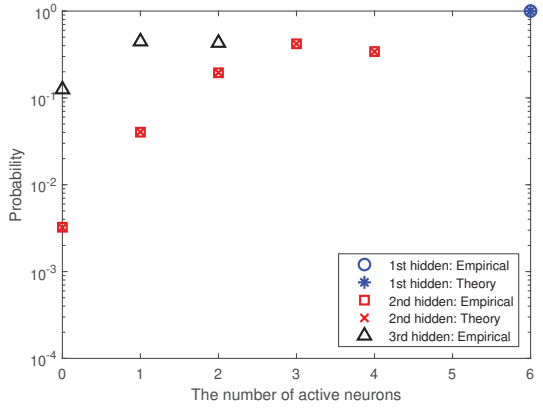
$$[\hat{P}_t]_{l,r} = \Pr(\mathfrak{m}_t = k_t^r, \mathfrak{d}_t^b = k_{t,b}^r | \mathfrak{m}_s = k_s^l, \mathfrak{d}_s^b = k_{s,b}^l, \forall 1 \le s < t) \prod_{j=1}^{t-1} \delta_{k_j^l = k_j^r} \delta_{k_{j,b}^l = k_{j,b}^r}. \quad \text{(F.1)}$$



(a)



(b)



(c)

FIG. E1: The probability distributions of the number of active neurons at different layers are shown for a ReLU network having $\boldsymbol{n} = (1, 6, 4, 2, n_4, \cdots, n_L)$ architecture. (a) All layers are initialized by the unit hypersphere with bias. (b) All layers are initialized by the unit hypersphere without bias. (c) The first hidden layer is initialized by the unit hypersphere without bias. All other layers are initialized by the normal with bias.

For $t = L-1$, let $\hat{P}_{L-1}$ be a matrix of size $\prod_{j=1}^{L-2} s_j \times s_{L-1}$ such that for $l = 1, \ldots, \prod_{j=1}^{L-2} s_j$ and $r = 1, \ldots, s_{L-1}$,

$$[\hat{P}_{L-1}]_{l,r} = \Pr(\mathfrak{m}_{L-1} = \bar{k}^r_{L-1}, \mathfrak{d}^b_{L-1} = \bar{k}^r_{L-1,b} | \mathfrak{m}_s = k^l_s, \mathfrak{d}^b_s = k^l_{s,b}, \forall 1 \le s < L-2), \quad \text{(F.2)}$$

where $\bar{\mathrm{k}}^r_{L-1} = (\bar{k}^r_{L-1}, \bar{k}^r_{L-1,b})$ is the rth multi-index of the lexicographic ordering of $[\hat{n}_{L-1}] \times [\hat{m}_{L-1}]$.

Once the above stochastic matrices are all identified, its corresponding trainability readily follows based on the formulation given below.

Lemma 8. *For a learning task that requires a L-layer ReLU network having at least m_t active neurons in the tth layer, the trainability for a L-layer ReLU network with $\boldsymbol{n} = (n_0, n_1, \cdots, n_L)$ architecture is given as follow. Let $\tilde{n}_t = n_t - m_t + 1$. Then the trainability is given by*

$$\textit{Trainability } = \pi'_1 P'_2 \cdots P'_{L-1} 1_{\tilde{n}_{L-1}} + \pi'_1 \hat{P}_2 \cdots \hat{P}_{L-1} 1_{s_{L-1}},$$

where π'_1 is a $1 \times \tilde{n}_1$ submatrix of π_1 whose first component is $[\pi_1]_{m_t}$, P'_t is a $\tilde{n}_{t-1} \times \tilde{n}_t$ submatrix of P_t whose $(1,1)$ component is $[P_t]_{m_{t-1}, m_t}$, and 1_p is a $p \times 1$ vector whose entries are all 1's. Here π_1 and P_t are defined in Lemma 4, and $\{\hat{P}_t\}$ is defined in (F.1) and (F.2).

Proof of Lemma 8. We observe that

$$\Pr(\mathfrak{m}_t \ge 1, \mathfrak{d}^b_t \le n_t - m_t, \forall 1 \le t < L) = \Pr(\mathfrak{m}_t \ge m_t, \forall 1 \le t < L)$$
$$+ \Pr(\mathfrak{m}_1 \ge m_1, 1 \le \mathfrak{m}_t < m_t, \mathfrak{d}^b_t \le n_t - m_t, \forall 1 \le t < L).$$

From Lemma 4, it can be checked that

$$\Pr(\mathfrak{m}_t \ge m_t, \forall 1 \le t < L) = \pi'_1 \mathrm{P}'_2 \cdots \mathrm{P}'_{L-1} 1_{\tilde{n}_{L-1}}.$$

For convenience, let $\hat{\mathfrak{m}}_t = (\mathfrak{m}_t, \mathfrak{d}^b_t)$ for $t > 1$ and $\hat{\mathfrak{m}}_1 = \mathfrak{m}_t$. Let $\vec{\mathfrak{m}}_t = (\hat{\mathfrak{m}}_1, \hat{\mathfrak{m}}_2, \cdots, \hat{\mathfrak{m}}_t)$. Also, recall that $\mathcal{T}_{t-1} := [\hat{n}_1] \times [\hat{n}_2] \times [\hat{m}_2] \cdots \times [\hat{n}_{t-1}] \times [\hat{m}_{t-1}]$, where $[\hat{n}_t] = \{0, \cdots, n_t - m_t\}$ and $[\hat{m}_t] = \{1, \cdots, m_t - 1\}$. Let $\hat{T}_t := [\hat{n}_t] \times [\hat{m}_t]$. Also let $\vec{\pi}_t = [\Pr(\vec{\mathfrak{m}}_t = \mathrm{k})]_{\mathrm{k} \in \mathcal{T}_t}$ be the distribution of $\vec{\mathfrak{m}}_t$ restricted to $\mathcal{T}_t$. Then,

$$\begin{aligned}
&\Pr(\mathfrak{m}_1 \ge m_1, 1 \le \mathfrak{m}_t < m_t, \mathfrak{d}^b_t \le n_t - m_t, \forall 1 \le t < L) \\
&= \Pr(\vec{\mathfrak{m}}_{L-1} \in \mathcal{T}_{L-1}) = \sum_{\mathrm{k}_{L-1} \in \mathcal{T}_{L-1}} \Pr(\vec{\mathfrak{m}}_{L-1} = \mathrm{k}_{L-1}) \\
&= \sum_{\hat{\mathrm{k}}_{L-1} \in \hat{T}_{L-1}} \sum_{\mathrm{k}_{L-2} \in \mathcal{T}_{L-2}} \Pr(\hat{\mathfrak{m}}_{L-1} = \hat{\mathrm{k}}_{L-1} | \vec{\mathfrak{m}}_{L-2} = \mathrm{k}_{L-2}) \Pr(\vec{\mathfrak{m}}_{L-2} = \mathrm{k}_{L-2}) \\
&= \vec{\pi}_{L-2} \hat{P}_{L-1} 1_{s_{L-1}}.
\end{aligned}$$

It then suffices to identify $\vec{\pi}_t$ for $1 \le t < L-1$. Then note that for each $\mathrm{k}_t = (\mathrm{k}_{t-1}, \hat{\mathrm{k}}_t) \in \mathcal{T}_t$,

$$\Pr(\vec{\mathfrak{m}}_t = \mathrm{k}_t) = \Pr(\hat{\mathfrak{m}}_t = \hat{\mathrm{k}}_t | \vec{\mathfrak{m}}_{t-1} = \mathrm{k}_{t-1}) \Pr(\vec{\mathfrak{m}}_{t-1} = \mathrm{k}_{t-1}).$$

Thus we have $\vec{\pi}_t = \vec{\pi}_{t-1} \hat{P}_t$. Since $\vec{\pi}_1 = \pi'_1$, by recursively applying it, the proof is completed. □

We are now in a position to present our proof of Theorem 3.

Proof of Theorem 3. Given the event A_{t-1}^i that exactly i neurons are active in the $(t-1)$th hidden layer, let $\mathfrak{p}_{t,b}(A_{t-1}^i)$ and $\mathfrak{p}_{t,g}(A_{t-1}^i)$ be the conditional probabilities that a neuron in the tth hidden layer is born dead permanently and born dead tentatively, respectively. Then

$$\mathfrak{p}_t(A_{t-1}^i) = \mathfrak{p}_{t,b}(A_{t-1}^i) + \mathfrak{p}_{t,g}(A_{t-1}^i).$$

Note that since the weights and the biases are initialized from a symmetric probability distribution around 0, we have $\mathfrak{p}_{t,b}(A_{t-1}^i) \geq 2^{-i-1}$. This happens when all the weights and bias are initialized to be nonpositive. Let $\mathfrak{d}_t^g$ and $\mathfrak{d}_t^b$ be the number of tentatively dead and permanently dead neurons at the tth hidden layer. It then can be checked that

$$\begin{aligned}
&\Pr(\mathfrak{d}_t^g = j_1, \mathfrak{d}_t^b = j_2 | \mathfrak{m}_1 = i) \\
&\quad = \binom{n_t}{j_1, j_2, j_3} \mathbb{E}_{t-1} \left\{ [1 - \mathfrak{p}_t(A_{t-1}^i)]^{n_t - j_1 - j_2} [\mathfrak{p}_{t,g}(A_{t-1}^i)]^{j_1} [\mathfrak{p}_{t,b}(A_{t-1}^i)]^{j_2} \right\},
\end{aligned}$$

where $j_3 = n_t - j_1 - j_2$, $\mathbb{E}_{t-1}$ is the expectation with respect to $\mathcal{F}_{t-1}$, and

$$\binom{n}{k_1, k_2, k_3}$$

is a multinomial coefficient. Also note that $\mathfrak{m}_t + \mathfrak{d}_t^g + \mathfrak{d}_t^b = n_t$. It then follows from Lemma 8 that

$$\begin{aligned}
&\Pr(\mathfrak{m}_t \geq 1, \mathfrak{d}_t^b \leq n_t - m_t, \forall 1 \leq t < 3) \\
&\quad = \Pr(\mathfrak{m}_1 \geq m_1, \mathfrak{m}_2 \geq m_2) + \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \sum_{k=m_1}^{n_1} \Pr(\mathfrak{m}_2 = j, \mathfrak{d}_2^b = l | \mathfrak{m}_1 = k) \\
&\quad \times \Pr(\mathfrak{m}_1 = k) = \Pr(\mathfrak{m}_1 \geq m_1, \mathfrak{m}_2 \geq m_2) + \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \sum_{k=m_1}^{n_1} \Pr \\
&\quad \times (\mathfrak{d}_2^g = n_2 - j - l, \mathfrak{d}_2^b = l | \mathfrak{m}_1 = k) \Pr(\mathfrak{m}_1 = k) = \Pr(\mathfrak{m}_1 \geq m_1, \mathfrak{m}_2 \geq m_2) \\
&\quad + \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \sum_{k=m_1}^{n_1} \binom{n_2}{n_2 - j - l, j, l} \mathbb{E}_1 \left[(1 - \mathfrak{p}_2(A_1^k))^j (\mathfrak{p}_{2,g}(A_1^k))^{n_2 - j - l} (\mathfrak{p}_{2,b}(A_1^k))^l \right] \\
&\quad \times \Pr(\mathfrak{m}_1 = k) \geq \Pr(\mathfrak{m}_1 \geq m_1, \mathfrak{m}_2 \geq m_2) + \sum_{j=1}^{m_2-1} \sum_{l=0}^{n_2-m_2} \sum_{k=m_1}^{n_1} \binom{n_2}{n_2 - j - l, j, l} \\
&\quad \times \mathbb{E}_1 \left[(1 - \mathfrak{p}_2(A_1^k))^j (\mathfrak{p}_2(A_1^k) - 2^{-k-1})^{n_2 - j - l} (2^{-k-1})^l \right] \Pr(\mathfrak{m}_1 = k).
\end{aligned}$$

Since $\mathfrak{p}_2(A_1^k)$ is identified by Lemma 6 and $\Pr(\mathfrak{m}_1 = k)$ is identified by Lemma 5, by plugging it into the above, the proof is completed. □

APPENDIX G. PROOF OF COROLLARY 1

Proof. Note that

$$\Pr(\mathfrak{m}_t \geq 1, \mathfrak{d}_t^b \leq n_t - m_t, \forall t = 1, \cdots, L) \leq \Pr(\mathfrak{m}_t \geq 1, \forall t = 1, \cdots, L),$$

and

$$1 - \Pr(\mathfrak{m}_t \geq 1, \forall t = 1, \cdots, L) = \Pr(\exists t, \text{ such that } \mathfrak{m}_t = 0) = \Pr(\mathcal{N}^{L+1}(\mathbf{x}) \text{ is born dead}).$$

It was shown in Theorem 3 of Lu et al. (2019) that

$$\Pr(\mathcal{N}^L(\mathbf{x}) \text{ is born dead}) \geq 1 - \mathfrak{a}_1^{L-2} + \frac{(1-2^{-n+1})(1-2^{-n})}{1+(n-1)2^{-n}}(-\mathfrak{a}_1^{L-2} + \mathfrak{a}_2^{L-2}),$$

where $\mathfrak{a}_1 = 1 - 2^{-n}$ and $\mathfrak{a}_2 = 1 - 2^{-n+1} - (n-1)2^{-2n}$. Thus the proof is completed. □

APPENDIX H. PROOF OF THEOREM 4

Proof. Since $q(\mathbf{x}) = E[\|\mathcal{N}(\mathbf{x})\|^2]/d_{\text{out}}$ and the rows of $\boldsymbol{W}^2$ are independent, without loss of generality let us assume $d_{\text{out}} = 1$. The direct calculation shows that

$$E[\|\mathcal{N}(\mathbf{x}_k)\|^2] = \sum_{i=1}^{N} \sigma_{\text{out}}^2 E\left[\phi(\boldsymbol{w}_i^T \mathbf{x}_k + \boldsymbol{b}_i)^2\right] = \frac{N\sigma_{\text{out}}^2}{N_{\text{train}}} \left\{ \sum_{i=1}^{N_{\text{train}}} E\left[\phi(\boldsymbol{w}_i^T(\mathbf{x}_k - \mathbf{x}_i) + |\epsilon_i|)^2\right] \right\}.$$

Let $\sigma_{k,i}^2 = \sigma_{\text{in}}^2 \|\mathbf{x}_k - \mathbf{x}_i\|^2$ and $\epsilon_{k,i} = |\epsilon_i|/\sigma_{k,i}$. Note that $\boldsymbol{w}_i^T(\mathbf{x}_k - x_i) \sim N(0, \sigma_{k,i}^2)$. Then

$$E\left[\phi(\boldsymbol{w}_i^T(\mathbf{x}_k - \mathbf{x}_i) + |\epsilon_i|)^2 | \epsilon_i\right] = I_1(\epsilon_i) + I_2(\epsilon_i),$$

where

$$I_1(\epsilon) = \int_0^\infty (z+\epsilon)^2 \frac{e^{-z^2/(2\sigma_{k,i}^2)}}{\sqrt{2\pi\sigma_{k,i}^2}} dz, \quad I_2(\epsilon) = \int_{-\epsilon}^0 (z+\epsilon)^2 \frac{e^{-z^2/(2\sigma_{k,i}^2)}}{\sqrt{2\pi\sigma_{k,i}^2}} dz.$$

Then if $\epsilon_i = |e_i|$ where $e_i \sim \text{N}(0, \sigma_{e,i}^2)$, we have

$$I_1(\epsilon_i) = \frac{1}{2}\sigma_{k,i}^2 + \sqrt{\frac{2}{\pi}}\sigma_{k,i}\epsilon_i + \frac{1}{2}\epsilon_i^2 \implies E[I_1(\epsilon_i)] = \frac{1}{2}\sigma_{k,i}^2 + \frac{2}{\pi}\sigma_{k,i}\sigma_{e,i} + \frac{1}{2}\sigma_{e,i}^2.$$

Also, we have

$$I_2(\epsilon) = \int_{-\epsilon}^0 (z+\epsilon)^2 \frac{e^{-z^2/(2\sigma_{k,i}^2)}}{\sqrt{2\pi\sigma_{k,i}^2}} dz = \sigma_{k,i}^2 \int_{-\epsilon_{k,i}}^0 (z+\epsilon_{k,i})^2 \frac{e^{-z^2/2}}{\sqrt{2\pi}} dz$$

$$= \sigma_{k,i}^2 \left[\frac{1}{2}(\epsilon_{k,i}^2 + 1)\text{erf}\left(\frac{\epsilon_{k,i}}{\sqrt{2}}\right) + \frac{\epsilon_{k,i}(e^{-\epsilon_{k,i}^2/2} - 2)}{\sqrt{2\pi}} \right],$$

where $\epsilon_{k,i} = |e_{k,i}|$ and $e_{k,i} \sim \mathrm{N}(0, \sigma_{e,i}^2/\sigma_{k,i}^2)$. Note that if $z = |z'|$ where $z' \sim \mathrm{N}(0, \sigma^2)$,

$$E[z^2 \mathrm{erf}(z/\sqrt{2})] = \frac{2\sigma^2 \tan^{-1}(\sigma)}{\pi} + \frac{2\sigma^3}{\pi(\sigma^2+1)},$$

$$E[\mathrm{erf}(z/\sqrt{2})] = \frac{2\tan^{-1}(\sigma)}{\pi}, \quad E[ze^{-z^2/2}] = \frac{2\sigma}{\sqrt{2\pi}(\sigma^2+1)}, \quad E[z] = \frac{2\sigma}{\sqrt{2\pi}}.$$

Therefore

$$E\left[\frac{1}{2}(z^2+1)\mathrm{erf}(z/\sqrt{2}) + \frac{ze^{-z^2/2} - 2z}{\sqrt{2\pi}}\right] = \frac{(\sigma^2+1)\tan^{-1}(\sigma)}{\pi} - \frac{\sigma}{\pi}.$$

By setting $\sigma = \sigma_{e,i}/\sigma_{k,i}$, we have

$$E[I_2(\epsilon_i)] = \sigma_{k,i}^2 \mathbb{E}_{\epsilon_{k,i}}\left[\frac{1}{2}(\epsilon_{k,i}^2+1)\mathrm{erf}\left(\frac{\epsilon_{k,i}}{\sqrt{2}}\right) + \frac{\epsilon_{k,i}(e^{-\epsilon_{k,i}^2/2} - 2)}{\sqrt{2\pi}}\right],$$

$$= \frac{(\sigma_{e,i}^2 + \sigma_{k,i}^2)\tan^{-1}(\sigma_{e,i}/\sigma_{k,i})}{\pi} - \frac{\sigma_{e,i}\sigma_{k,i}}{\pi} := \gamma_i.$$

Thus we have

$$E\left[\phi(\boldsymbol{w}_i^T(\mathbf{x}_k - \mathbf{x}_i) + |\epsilon_i|)^2\right] = E[I_1(\epsilon_i)] + E[I_2(\epsilon_i)] = \frac{1}{2}\sigma_{k,i}^2 + \frac{2}{\pi}\sigma_{k,i}\sigma_{e,i} + \frac{1}{2}\sigma_{e,i}^2 + \gamma_i,$$

and thus

$$E[\|\mathcal{N}(\mathbf{x}_k)\|^2] = \frac{N\sigma_{\text{out}}^2}{N_{\text{train}}}\sum_{i=1}^{N_{\text{train}}}\left[\frac{1}{2}\sigma_{k,i}^2 + \frac{1}{2}\sigma_{e,i}^2 + \frac{2}{\pi}\sigma_{k,i}\sigma_{e,i} + \gamma_i\right].$$

Let $\sigma_{e,i}^2 = \sigma_e^2 = \sigma_{\text{in}}^2 s^2$ for all i. Then we have

$$E[q(\mathbf{x}_k)] = E[\|\mathcal{N}(\mathbf{x}_k)\|^2] = \frac{N\sigma_{\text{out}}^2\sigma_{\text{in}}^2}{N_{\text{train}}\pi}\sum_{i=1}^{N_{\text{train}}}\left[(s^2 + \Delta_{k,i}^2)\left(\tan^{-1}(s/\Delta_{k,i}) + \pi/2\right) + s\Delta_{k,i}\right],$$

where $\Delta_{k,i} = \|\mathbf{x}_k - \mathbf{x}_i\|_2$. Thus, we obtain

$$\mathbb{E}_{\mathcal{X}_m}[q(\mathbf{x})] = \frac{1}{N_{\text{Ntrain}}}\sum_{k=1}^{N_{\text{train}}} E[q(\mathbf{x}_k)] = \frac{N\sigma_{\text{out}}^2\sigma_{\text{in}}^2}{N_{\text{train}}^2\pi}\sum_{k,i=1}^{N_{\text{train}}}$$
$$\times\left[(s^2 + \Delta_{k,i}^2)\left(\tan^{-1}(s/\Delta_{k,i}) + \pi/2\right) + s\Delta_{k,i}\right],$$

which completes the proof. □

Journal of Machine Learning for Modeling and Computing, 1(1):75–95 (2020)

MACHINE LEARNING FOR TRAJECTORIES OF PARAMETRIC NONLINEAR DYNAMICAL SYSTEMS

Roland Pulch* & Maha Youssef

Institute for Mathematics and Computer Science, University of Greifswald, Walther-Rathenau-Str. 47, D-17489 Greifswald, Germany

*Address all correspondence to: Roland Pulch, Institute for Mathematics and Computer Science, University of Greifswald, Walther-Rathenau-Str. 47, D-17489 Greifswald, Germany, E-mail: roland.pulch@uni-greifswald.de

Original Manuscript Submitted: 3/5/2020; Final Draft Received: 6/18/2020

We investigate parameter-dependent nonlinear dynamical systems consisting of ordinary differential equations or differential-algebraic equations. A single quantity of interest is observed, which depends on the solution of a system. Our aim is to determine efficient approximations of the trajectories belonging to the quantity of interest in the time domain. We arrange a set of samples including trajectories of this quantity. A proper orthogonal decomposition of this data yields a reduced basis. Consequently, the mapping from the parameter domain to the basis coefficients is approximated. We apply machine learning with artificial neural networks for this approximation, where the degrees of freedom are fitted to the data of the sample trajectories in a nonlinear optimization. Alternatively, we consider a polynomial approximation, which is identified by regression, for comparison. Furthermore, concepts of sensitivity analysis are examined to characterize the impact of an input parameter on the output of the exact mapping or the approximations from the neural networks. We present results of numerical computations for examples of nonlinear dynamical systems.

KEY WORDS: *nonlinear dynamical system, differential-algebraic equation, initial value problem, parametric model order reduction, proper orthogonal decomposition, machine learning, neural network, polynomial regression, sensitivity analysis*

1. INTRODUCTION

Mathematical modeling of real-world problems often yields dynamical systems in science and engineering. We consider initial value problems for nonlinear systems of ordinary differential equations (ODEs) or differential-algebraic equations (DAEs), which depend on physical parameters. A transient quantity of interest (QoI) is defined depending on the solution of a system. Many evaluations of the QoI are required for different realizations of the parameters in some tasks like optimization and uncertainty quantification; see Xiu (2010), for example. Often a time integration of the dynamical system is costly for realistic applications. Thus our aim is to determine efficient approximations of the trajectories associated with the parameter-dependent QoI. An evaluation of this approximation should be cheap, while good accuracy is still achieved for most of the relevant parameter values.

We determine the trajectories of the QoI for parameter samples in some bounded parameter domain. Proper orthogonal decomposition (POD) represents a method for projection-based

2689–3967/20/$35.00

model order reduction (Antoulas, 2005; Kunisch and Volkwein, 2001). We use a similar POD approach to obtain a reduced basis for the trajectories. Any trajectory is approximated by a function in a low-dimensional space. The approximation is uniquely determined by the coefficients in the reduced basis. Hence we obtain a mapping between finite-dimensional spaces, where a parameter value is mapped to the basis coefficients.

Now the task is to determine an efficient approximation of the mapping between finite-dimensional spaces. This procedure can be seen as a kind of parametric model order reduction (pMOR) (Benner et al., 2015). Instead of solving the full-order model consisting of the dynamical system, a parameter-dependent reduced-order model is constructed. However, our reduced-order model is not a dynamical system anymore. On the one hand, we apply a concept of machine learning based on artificial neural networks (Du and Swamy, 2014; Goodfellow et al., 2017). On the other hand, we use a multivariate polynomial regression for comparison (Seber and Lee, 2003). In both approaches, an optimization process is performed to identify the degrees of freedom appropriately, where the data of the sample trajectories is included. This optimization is called training in the case of machine learning. Neural networks (NNs) imply a nonlinear optimization problem, whereas the polynomial fit requires just the solution of linear least squares problems. Both approaches represent data-driven methods.

Similar problems have also been tackled by NNs in previous works. The POD method was used for parametric stationary solutions of partial differential equations in Hesthaven and Ubbiali (2018) and Yu and Hesthaven (2019). Trajectories of solutions satisfying (non-parametric) autonomous systems of ODEs were reproduced by Qin et al. (2019).

Furthermore, we discuss a variance-based sensitivity analysis of the input–output behavior in the mapping between the finite-dimensional spaces. The total effect sensitivity indices yield a quantification of the impact of the individual parameters (Saltelli et al., 2008; Sobol and Kucherenko, 2009). Thus a ranking of the importance is feasible for the parameters. The variance-based sensitivity analysis can be performed for both the exact mapping and an approximation. Alternatively, we also investigate the weights in a trained NN to obtain information about the sensitivities with respect to the input parameters.

We present numerical results for two examples, which are nonlinear systems of DAEs modeling electric circuits. Both the machine learning approach and the polynomial regression are used to obtain the approximations. The errors of the methods are analyzed and compared. In addition, we illustrate the sensitivity analysis by the examples.

In this article, we introduce parametric nonlinear dynamical systems and the investigated problem in Section 2. The POD method yields the representation in the reduced basis, and the polynomial approximation is outlined. The variance-based sensitivity indices are formulated for our problem. In Section 3, we apply artificial NNs for the approximation. We define the weight-based sensitivity measures. The sources of errors are discussed for the entire numerical method. Finally, Section 4 demonstrates results of numerical computations for the two examples.

2. PROBLEM DEFINITION

We describe the problem in this section, which will be tackled by artificial NNs.

2.1 Nonlinear Dynamical Systems

We consider nonlinear dynamical systems in the form

$$\begin{aligned} M(p)\dot{x}(t,p) &= f(t,x(t,p),p) \\ y(t,p) &= g(x(t,p),p). \end{aligned} \tag{1}$$

The mass matrix M and/or the right-hand side f depend on physical parameters $p \in \Pi \subseteq \mathbb{R}^q$. Hence the state variables or inner variables $x : [t_0, t_{\mathrm{end}}] \times \Pi \to \mathbb{R}^n$ depend on time as well as the parameters. If the mass matrix is non-singular, then we obtain a system of ODEs. In contrast, a singular mass matrix implies a system of DAEs. Initial value problems (IVPs) are specified by

$$x(t_0, p) = x_0(p), \tag{2}$$

with a predetermined function $x_0 : \Pi \to \mathbb{R}^n$. In the case of DAEs, the initial values have to be consistent. The consistency conditions represent systems of algebraic equations. Consistent initial values typically depend on the physical parameters of the system.

A QoI y: $[t_0, t_{\mathrm{end}}] \times \Pi \to \mathbb{R}^{n_{\mathrm{qoi}}}$ is defined by the function g depending on the solution x of the dynamical system [Eq. (1)]. We assume that a single QoI ($n_{\mathrm{qoi}} = 1$) is under investigation. Often y depends linearly on the variables x. Sometimes y coincides with a single component of x.

We suppose that each parameter is located in a compact interval: $p_j \in [p_{j,\min}, p_{j,\max}]$ for $j = 1, \ldots, q$. Consequently, the parameter domain is a multidimensional cuboid. Without loss of generality, we assume that the parameter domain is the unit hypercube $\mathcal{H}_q = [0,1]^q$. In this standardization, the bijective mapping reads as

$$\Xi : \mathcal{H}_q \to \Pi, \quad p_j \mapsto p_{j,\min}(1 - p_j) + p_{j,\max} p_j \quad \text{for } j = 1, \ldots, q,$$

where Π is the multidimensional cuboid incorporating the physical quantities.

The following strategy can be applied for boundary value problems (BVPs) of dynamical systems as well, because only the information of the trajectories of the QoI is included. It does not matter if the trajectories are computed by IVPs or BVPs. The techniques are data-driven.

2.2 Proper Orthogonal Decomposition

In Mifsud et al. (2016) POD was used for ensembles of solutions at different parameter values. We employ this idea for the transient problems [Eq. (1)]. A set of parameter samples

$$\mathcal{S} = \{p_1, \ldots, p_k\} \subset \mathcal{H}_q \tag{3}$$

is generated. For example, random samples can be chosen in $\mathcal{H}_q$.

A discretization in time implies a grid with points $t_1, \ldots, t_m$ satisfying $t_0 \le t_1 < t_2 < \cdots < t_m \le t_{\mathrm{end}}$. The initial point t_0 may be included in the grid. Consequently, a time integration of the IVPs [Eqs. (1) and (2)] yields the values of the QoI in the time points. We assume that the errors of the time integration are negligible. Let $Y \in \mathbb{R}^{m \times k}$ be the matrix with entries $y_{ij} := y(t_i, p_j)$, which represent discrete observations at the parameter samples [Eq. (3)]. We perform a POD by the singular value decomposition

$$Y = USV^\top, \tag{4}$$

with a diagonal matrix $S \in \mathbb{R}^{m \times k}$ containing the singular values $\sigma_1, \sigma_2, \ldots, \sigma_s$ in descending order with $s = \min\{m, k\}$. The orthogonal matrix $U \in \mathbb{R}^{m \times m}$ includes the associated basis

vectors $u_1, \ldots, u_m$ in its columns. Taking the r dominant singular values, we form the smaller matrix $\tilde{U} \in \mathbb{R}^{m \times r}$ with the columns $u_1, \ldots, u_r$. Let $a = (a_1, \ldots, a_r)^\top \in \mathbb{R}^r$ be coefficients. If $y(\cdot, p)$ is a trajectory of the QoI for any $p \in \mathcal{H}_q$, then its representation in the reduced basis reads as

$$w(p) := \begin{pmatrix} y(t_1, p) \\ \vdots \\ y(t_m, p) \end{pmatrix} \approx \begin{pmatrix} \tilde{y}(t_1, p) \\ \vdots \\ \tilde{y}(t_m, p) \end{pmatrix} := \sum_{\ell=1}^{r} a_\ell u_\ell = \tilde{U} a. \tag{5}$$

The best approximation of the coefficients a is obtained by the projection

$$\hat{a}(p) = \tilde{U}^\top w(p). \tag{6}$$

The accuracy of the POD is characterized by the requirement

$$\sum_{\ell=1}^{r} \sigma_\ell^2 > \delta \sum_{\ell=1}^{s} \sigma_\ell^2, \tag{7}$$

including a user-specified tolerance δ, say $\delta \geq 0.999$ (Benner et al., 2015, p. 502). The smallest rank r is chosen such that the condition [Eq. (7)] is satisfied.

If the coefficients are given, then the right-hand side of Eq. (5) implies an approximation of the transient QoI. The time integration produces the transient QoI in the left-hand side of Eq. (5) and thus the projection [Eq. (6)] yields the mapping

$$\Gamma : \mathcal{H}_q \to \mathbb{R}^r, \quad p \mapsto \hat{a}. \tag{8}$$

We want to approximate this nonlinear function between low-dimensional spaces.

2.3 Polynomial Regression

For comparison, we arrange a straightforward polynomial approximation of the mapping [Eq. (8)]. We apply the Legendre polynomials as basis functions (Stoer and Bulirsch, 2002, p. 177) because well-conditioned problems are expected in comparison to other bases like the monomial basis, for example. The polynomial approximation reads as $\tilde{a} : \mathcal{H}_q \to \mathbb{R}^r$ with

$$\tilde{a}(p) = \sum_{i=1}^{s} c_i \Phi_i(p), \tag{9}$$

including vectors $c_i = (\gamma_{i1}, \ldots, \gamma_{ir})^\top \in \mathbb{R}^r$. The multivariate basis polynomials are the products of the (univariate) Legendre polynomials

$$\Phi_i(p) = L_{i_1}(p_1) L_{i_2}(p_2) \cdots L_{i_q}(p_q), \tag{10}$$

for $i = 1, \ldots, s$ with $p = (p_1, p_2, \ldots, p_q)^\top$. There is a one-to-one mapping from the integers i to the multiindices $(i_1, \ldots, i_q)$. The traditional Legendre polynomials are linearly transformed from their domain of dependence $[-1, 1]$ to $[0, 1]$. The degree of $L_j : [0, 1] \to \mathbb{R}$ is exactly j. Hence the total degree of a multivariate polynomial [Eq. (10)] is $i_1 + \cdots + i_q$. The number of basis polynomials up to a total degree d is (Xiu, 2010, p. 65),

$$s = \frac{(q + d)!}{q! d!}. \tag{11}$$

Now we employ the set [Eq. (3)] consisting of k parameter samples, which was also used for the computation of the reduced basis in the POD method. Let $s < k$. We arrange a Vandermonde matrix $V \in \mathbb{R}^{k \times s}$ and right-hand sides $b_\ell \in \mathbb{R}^k$ for $\ell = 1, \ldots, r$ by

$$V = \begin{pmatrix} \Phi_1(p^{(1)}) & \Phi_2(p^{(1)}) & \cdots & \Phi_s(p^{(1)}) \\ \Phi_1(p^{(2)}) & \Phi_2(p^{(2)}) & \cdots & \Phi_s(p^{(2)}) \\ \vdots & \vdots & & \vdots \\ \Phi_1(p^{(k)}) & \Phi_2(p^{(k)}) & \cdots & \Phi_s(p^{(k)}) \end{pmatrix} \quad \text{and} \quad b_\ell = \begin{pmatrix} \hat{a}_\ell(p^{(1)}) \\ \hat{a}_\ell(p^{(2)}) \\ \vdots \\ \hat{a}_\ell(p^{(k)}) \end{pmatrix}.$$

Now we solve the linear least squares problems

$$\min_{z_\ell \in \mathbb{R}^s} \|V z_\ell - b_\ell\|_2, \tag{12}$$

including the Euclidean norm $\|\cdot\|_2$. Each solution z_ℓ yields the coefficients $\gamma_{1\ell}, \ldots, \gamma_{s\ell}$ for $\ell = 1, \ldots, r$. Therein, a QR-decomposition (Golub and van Loan, 1996) of the matrix V can be reused for each right-hand side. Since this decomposition dominates the computational effort, the dimension r of the reduced basis is not significant. Consequently, the approximation [Eq. (9)] is identified. This approach is also called (multivariate) polynomial regression as in Seber and Lee (2003).

The polynomial regression may suffer from the effect of overfitting in the case of higher-degree polynomials. A regularization like Tikhonov's method or (discrete) $\mathcal{L}^2$-regularization can prevent overfitting (Wang, 2019). However, a similar approximation error often results by simply restricting to polynomials of lower degree.

Furthermore, we note that a polynomial approximation can be constructed using the trajectories of the samples in the discrete time points (without a reduced basis). Thus the approximation error of the POD method is avoided. The computation work does not become much larger than in our approach, since the matrix of the linear least squares problems is identical in all time points. Yet a separate polynomial occurs for each time point, which generates a large number of polynomials. In contrast, the number of polynomials is equal to the dimension of the reduced basis in our approach. Hence a more compact description of the problem is achieved.

2.4 Sensitivity Analysis

There are derivative-based sensitivity measures and variance-based sensitivity measures (Sobol and Kucherenko, 2009). We consider a variance-based approach. Let $\mathcal{H}_q = [0,1]^q$ be the unit hypercube again. Given a function $f : \mathcal{H}_q \to \mathbb{R}$, we assume that $f \in \mathcal{L}^2(\mathcal{H}_q)$. The total variance of f reads as

$$V(f) = \int_{\mathcal{H}_q} f(p)^2 \, \mathrm{d}p - \left(\int_{\mathcal{H}_q} f(p) \, \mathrm{d}p \right)^2. \tag{13}$$

A sensitivity analysis is obsolete in the case of $V(f) = 0$, because f becomes a constant function. Thus we assume that $V(f) > 0$. Variance-based sensitivity measures often require the computation of partial variances. We define the partial variances using polynomial chaos expansions (PCEs); see Sudret (2008) or Pulch and Narayan (2019). The function f exhibits the PCE

$$f(p) = \sum_{i=1}^{\infty} \hat{f}_i \Phi_i(p), \tag{14}$$

including the multivariate Legendre polynomials [Eq. (10)]; see Xiu (2010). The coefficients read as

$$\hat{f}_i = \langle f, \Phi_i \rangle = \int_{\mathcal{H}_q} f(p)\Phi_i(p)\ \mathrm{d}p, \tag{15}$$

using the inner product of the Hilbert space $\mathcal{L}^2(\mathcal{H}_q)$. The series [Eq. (14)] converges in the norm of $\mathcal{L}^2(\mathcal{H}_q)$. We define the index sets

$$I_j = \{i \in \mathbb{N} \ : \ \Phi_i \text{ is non-constant in } p_j\},$$

for $j = 1, \ldots, q$. Now the partial variances read as

$$V_j(f) = \sum_{i \in I_j} |\hat{f}_i|^2 \qquad \text{for } \ j = 1, \ldots, q. \tag{16}$$

An alternative formula of the same partial variances is given in Sobol (2001).

The total effect sensitivity indices are defined by

$$S_j^{\mathrm{T}}(f) = \frac{V_j(f)}{V(f)} \qquad \text{for } \ j = 1, \ldots, q, \tag{17}$$

using Eqs. (13) and (16). It follows that $0 \leq S_j^{\mathrm{T}} \leq 1$ for each j. The sensitivity indices [Eq. (17)] quantify the impact of each parameter on the variability of the function f.

In numerical methods, we have to replace the PCE [Eq. (14)] by approximations like Eq. (9). First, the series is truncated to a finite sum. Second, the coefficients in Eq. (15) are approximated. Since the inner products represent multivariate integrals, quadrature methods or cubature methods can be used.

We consider the mapping of Eq. (8): $\Gamma : \mathcal{H}_q \to \mathbb{R}^r, p \mapsto \hat{a}(p)$. The above sensitivity analysis is applicable to each component of Γ separately. Thus the sensitivity indices are $S_j^{\mathrm{T}}(\hat{a}_i)$ for $j = 1, \ldots, q$ and $i = 1, \ldots, r$, which form an array of rq quantities. However, the importance of the coefficients $\hat{a}_i$ decreases for increasing i due to the decay of the singular values in the decomposition [Eq. (4)]. Alternatively, we observe the sensitivity measures

$$S_j^{\mathrm{T}}\left(\sum_{i=1}^{r} \hat{a}_i^2\right) \qquad \text{for } \ j = 1, \ldots, q, \tag{18}$$

which allows for a more compact discussion. These sensitivity indices characterize the impacts of the parameters on the (Euclidean) norm of the low-dimensional representation.

Furthermore, an approximation $\tilde{\Gamma}$ of the mapping [Eq. (8)] can be used to compute the sensitivity indices with a low computational effort, because the evaluations of $\tilde{\Gamma}$ are cheaper than the evaluations of Γ. The approximations are obtained from either the above polynomial approach or an artificial NN.

3. MACHINE LEARNING

We employ a strategy of machine learning to solve the problem introduced in Section 2.

3.1 Artificial Neural Networks

We apply an artificial NN (Du and Swamy, 2014; Genzel and Kutyniok, 2019) to represent the input–output relation of the mapping [Eq. (8)]. A similar approach was used for spatial solutions of partial differential equations in Yu and Hesthaven (2019).

Figure 1 illustrates the schematic of an NN with three hidden layers. In the general case, let $L+1$ be the total number of layers and N_ℓ be the number of neurons in the ℓth layer for $\ell = 0, 1, \ldots, L$. Hence the number of hidden layers is $L-1$. The values N_0 and N_L denote the numbers of input neurons and output neurons, respectively. The mathematical modeling of an NN consists of a chain of operators

$$\Psi = T_L \circ \rho \circ T_{L-1} \circ \rho \circ T_{L-2} \circ \cdots \circ \rho \circ T_2 \circ \rho \circ T_1. \tag{19}$$

The operators $T_\ell : \mathbb{R}^{N_{\ell-1}} \to \mathbb{R}^{N_\ell}$ are affine-linear functions:

$$T_\ell(z) = A_\ell z + b_\ell,$$

with matrices $A_\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and vectors $b_\ell \in \mathbb{R}^{N_\ell}$. The entries of A_ℓ and b_ℓ are called weights and biases, respectively. The operator ρ represents a nonlinear activation function $\rho : \mathbb{R} \to \mathbb{R}$; for example, the hyperbolic tangent sigmoid function

$$\rho(x) = \frac{2}{1 + \mathrm{e}^{-2x}} - 1, \tag{20}$$

or the rectified linear unit (ReLU)

$$\rho(x) = \begin{cases} 0 & \text{for } x < 0, \\ x & \text{for } x \geq 0. \end{cases}$$

In Eq. (19), the function ρ is evaluated on a vector separately for each component. If the number of hidden layers is larger or equal to 3, then the model is called a deep NN (deep learning). Otherwise, the model represents a shallow NN.

In our application, there are q inputs given by a parameter tuple $p \in \mathcal{H}_q$. The r outputs are the coefficients a in Eq. (5) associated with the reduced basis. The degrees of freedom (DOFs) are the weights and biases $\Theta = (A_\ell, b_\ell)_{\ell=1}^{L}$ in the optimization problem. An appropriate choice is determined by a minimization of the distances $\Gamma(p_j) - \Psi(p_j)$ for realizations $p_j \in \mathcal{H}_q$ of the parameters. A norm or distance function, which quantifies these differences, is called a performance function in the context of NNs. Typical performance functions are the mean squared error or the mean absolute error.

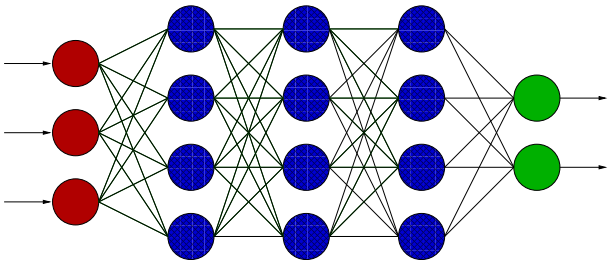


FIG. 1: Artificial neural network with input layer (left), hidden layers (center), and output layer (right)

In the determination of an NN, three sets of parameter samples

$$\mathcal{S}_{\text{train}} = \{p_1, \ldots, p_k\} \subset \mathcal{H}_q, \tag{21}$$

$$\mathcal{S}_{\text{valid}} = \{q_1, \ldots, q_{k'}\} \subset \mathcal{H}_q, \tag{22}$$

$$\mathcal{S}_{\text{test}} = \{r_1, \ldots, r_{k''}\} \subset \mathcal{H}_q, \tag{23}$$

are arranged, which are pairwise disjoint. The training set [Eq. (21)] is used to identify the DOFs in an iterative optimization method. Thus the performance function always decreases monotone for the training set in the iteration. The validation set [Eq. (22)] yields additional data to prevent an overfitting. The iteration is stopped if the performance function increases for the validation set. The test set [Eq. (23)] is not used in the optimization method at all. This independent set allows for an estimation of the accuracy achieved by a trained NN. In our application, we employ the set of samples [Eq. (3)], used in the POD method, also as the training set [Eq. (21)].

Concerning the context of pMOR, a technique consists of an offline phase and an online phase. In our offline phase, the sample trajectories are computed and an NN is trained. The computation of the trajectories involves significant computation work, since the nonlinear dynamical systems have to be solved. In our online phase, a trained NN is evaluated for possibly many parameter values, which is cheap.

3.2 Errors of the Methods

The approximations $\Psi(p) = \tilde{a}(p)$ from Eq. (19) imply the approximate trajectories $\tilde{y}(t_i, p)$ for each realization p of the parameters in Eq. (5). The total error consists of three parts:

(1) The numerical error of the time integration,

(2) The approximation error with respect to the reduced basis from POD,

(3) The approximation error of the NN.

We impose high accuracy requirements in the numerical time integration. Consequently, the error of part (1) becomes negligible. Although a tolerance $\delta \approx 1$ is applied in the condition [Eq. (7)], the error of part (2) may be relatively large for some parameter values if the associated trajectory is significantly different from the sample trajectories. Hence a decline of the error within part (2) also requires an increase in the number of samples in the POD. In the alternative approach of Section 2.3, just part (3) changes into the error of the polynomial approximation.

We estimate the error by a discrete $\mathcal{L}^1$-norm in time. Given a parameter tuple $p \in \mathcal{H}_q$, this error reads as

$$E(p) = \frac{1}{t_{\text{end}} - t_0} \sum_{i=1}^{m-1} (t_{i+1} - t_i) \, |y(t_i, p) - \tilde{y}(t_i, p)| \,, \tag{24}$$

assuming $t_0 = t_1$ and $t_{\text{end}} = t_m$. Our reference values $y(t_i, p)$ will still include an error of a numerical time integration. However, this error is negligible due to the high accuracy of the time integration. In the case of sample sets [Eqs. (21)–(23)], we observe statistics of the errors [Eq. (24)] like the mean value and the sample variance, for example.

3.3 Sensitivity Analysis Using Weights

In the field of machine learning and NNs, there is a sensitivity analysis based on a layer-wise relevance propagation and associated relevance scores (Montavon et al., 2018). However, the relevance scores depend on the inputs of the NN; that is different input values imply different relevance scores. In contrast, the variance-based sensitivity indices illustrated in Section 2.4 represent global sensitivity measures, which are defined for the complete parameter domain. Thus we examine the variance-based sensitivity analysis for our problem.

If an NN represents a good approximation of the mapping [Eq. (8)] meaning that $\Gamma(p) \approx \Psi(p)$ for all p, then the total effect sensitivity indices also agree for the mappings Γ and Ψ. We investigate the magnitudes of the weights in a trained NN to obtain an alternative sensitivity analysis. In the NN model [Eq. (19)], the first operator T_1 describes the mapping from the input layer to the first hidden layer. It holds that $T_1(z) = A_1 z + b_1$ with matrix $A_1 \in \mathbb{R}^{N_1 \times q}$ and vector $b_1 \in \mathbb{R}^{N_1}$. Let w_{ij} for $i = 1, \ldots, N_1$ and $j = 1, \ldots, q$ be the weights in A_1. We define sensitivity measures by the Euclidean norm of the set of weights associated to the jth input:

$$S_j^{\mathrm{W}} = \sum_{i=1}^{N_1} w_{ij}^2 \qquad \text{for } \; j = 1, \ldots, q. \tag{25}$$

The square of the Euclidean norm is used for a comparison to the variance-based sensitivity analysis, because the partial variances [Eq. (16)] are sums of squares. We do not consider the weights involved in the subsequent hidden layers, because such a weight cannot be assigned to a specific input any more.

In general, the number of neurons in a hidden layer is often chosen larger than the number of input neurons. Thus it holds that $N_1 > q$. Numerical computations of test examples show that a sensitivity coefficient [Eq. (25)] may not be small, even though the influence of the associated parameter on the outputs is insignificant. It follows that the first hidden layer gets input from a insignificant parameter, which is averaged out or canceled out in the subsequent layers.

We propose an approach to avoid this behavior. Assume that an NN is sufficiently accurate with $L-1$ hidden layers of sizes $N_1, \ldots, N_{L-1}$. We extend this network to L hidden layers with sizes $\hat{N}_1, \ldots, \hat{N}_L$ using $\hat{N}_1 = N_0$, $\hat{N}_\ell = N_{\ell-1}$ for $\ell = 2, \ldots, L$. The activation function between the input layer and the first hidden layer is chosen purely linear ($\rho(x) = x$ for all x); that is the identity operator. This extended NN reads as

$$\hat{\Psi} = \hat{T}_{L+1} \circ \rho \circ \hat{T}_L \circ \rho \circ \hat{T}_{L-1} \circ \cdots \circ \rho \circ \hat{T}_3 \circ \rho \circ \hat{T}_2 \circ \hat{T}_1. \tag{26}$$

The approximation quality of the extended NN is at least as good as that in the original NN, because choosing $\hat{T}_1$ as the identity and $\hat{T}_\ell = T_{\ell-1}$ for $\ell = 2, \ldots, L+1$ implies $\Psi = \hat{\Psi}$. However, the input information is not spread around a larger number of neurons in the first hidden layer. Hence this approach enforces a compact propagation of the information from the inputs. Now the sensitivity measures [Eq. (25)] are investigated for the extended NN [Eq. (26)], where it holds that $\hat{N}_1 = N_0 = q$.

The concept of the sensitivity measures [Eq. (25)] is heuristic. We will investigate the computed sensitivity indicators for the examples in Section 4. In particular, a comparison between Eqs. (17) and (25) is presented.

4. NUMERICAL RESULTS

We investigate two examples of nonlinear dynamical systems. All computations were performed on a FUJITSU Esprimo P920 Intel(R) Core(TM) i5-4570 CPU with 3.20 GHz (4 cores) and the Microsoft Windows 10 operating system. The software package MATLAB (version 9.7.0.1190202/R2019b) produced the numerical results. The NNs were trained using its deep learning toolbox.

4.1 Example: Transistor Amplifier

We consider the electric circuit of a transistor amplifier shown in Fig. 2. This circuit includes three capacitances, six resistances, and a bipolar transistor. In Hairer and Wanner (1996), a mathematical model is given, which consists of five DAEs for five unknown node voltages $u_1, \ldots, u_5$ ($n = 5$). The mass matrix and the right-hand side of Eq. (1) read as

$$M = \begin{pmatrix} -C_1 & C_1 & 0 & 0 & 0 \\ C_1 & -C_1 & 0 & 0 & 0 \\ 0 & 0 & -C_2 & 0 & 0 \\ 0 & 0 & 0 & -C_3 & C_3 \\ 0 & 0 & 0 & C_3 & -C_3 \end{pmatrix}$$

$$f = \begin{pmatrix} \dfrac{u_1}{R_0} \\ u_2\left(\dfrac{1}{R_1} + \dfrac{1}{R_2}\right) + (1-\gamma) w(u_2 - u_3) \\ \dfrac{u_3}{R_3} - w(u_2 - u_3) \\ \dfrac{u_4}{R_4} + \gamma w(u_2 - u_3) \\ \dfrac{u_5}{R_5} \end{pmatrix} + \begin{pmatrix} -\dfrac{u_{\mathrm{in}}}{R_0} \\ -\dfrac{u_{\mathrm{op}}}{R_2} \\ 0 \\ -\dfrac{u_{\mathrm{op}}}{R_4} \\ 0 \end{pmatrix}.$$

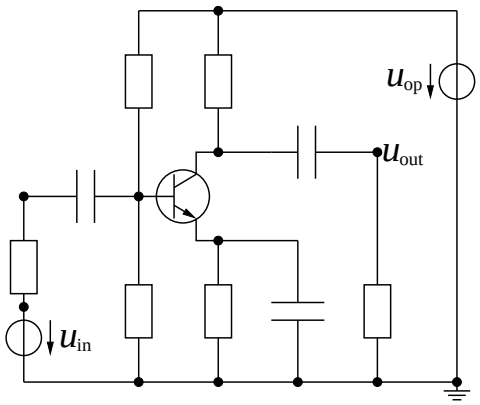


FIG. 2: Electric circuit of a transistor amplifier

The current-voltage relation of the bipolar transistor is described by the nonlinear function

$$w(u) = \alpha \left[\exp\left(\frac{u}{\beta}\right) - 1 \right], \tag{27}$$

with constants $\alpha = 10^{-6}$, $\beta = 0.026$, and $\gamma = 0.99$. We use nominal parameter values as given in Hairer and Wanner (1996): capacitances $C_1 = 10^{-6}$, $C_2 = 2 \cdot 10^{-6}$, and $C_3 = 3 \cdot 10^{-6}$; resitances $R_0 = 1000$, $R_1 = \cdots = R_5 = 9000$; and operating voltage $u_{\mathrm{op}} = 6$. The differential index of the system is one. We supply a harmonic oscillation

$$u_{\mathrm{in}}(t) = A \sin\left(\frac{2\pi}{T} t\right), \tag{28}$$

with period $T = 0.01$ and amplitude $A = 0.4$ as input voltage. The output voltage $u_{\mathrm{out}} = u_5$ represents the QoI.

We consider parameter variations in capacitances, resistances, and operating voltage. Variability of the parameters within the transistor model is not examined. Thus the dimension of the parameter domain is $q = 10$. Parameter variations of this example were also investigated for another purpose in Pulch (2019). A variation of 20% around the above nominal value is set for each parameter, which forms the multidimensional cuboid $\Pi \subset \mathbb{R}^{10}$.

Concerning the numerical solution of IVPs, we use the function `ode15s` in MATLAB, which is a multistep method based on the numerical differentiation formula (NDF; Shampine and Reichelt, 1997). We specify the same initial condition as a starting value for all parameters and the method determines consistent initial values [Eq. (2)] depending on the parameters. The time integrations are performed in the interval $[t_0, t_{\mathrm{end}}] = [0, 0.03]$ with local error control using relative tolerance $\varepsilon_{\mathrm{r}} = 10^{-6}$ and absolute tolerance $\varepsilon_{\mathrm{a}} = 10^{-8}$.

We produce the sets [Eqs. (21)–(23)] with $k = k' = k'' = 1000$ samples using pseudo random numbers in $\mathcal{H}_{10}$. The QoI is obtained in $m = 500$ equidistant points in the time interval $[t_0, t_{\mathrm{end}}]$ including t_0 and t_{end}, where the accuracy of the output agrees to the predetermined tolerances. Figure 3 illustrates both the trajectory for the mean values of the parameters and several trajectories for different parameter samples.

In the POD method, we apply the training set [Eq. (21)] only. The computed singular values are shown in Fig. 4. We observe a fast decay of the singular values. The reduced dimension $r = 9$ is the smallest number satisfying the accuracy requirement [Eq. (7)] for the threshold $\delta = 0.999$.

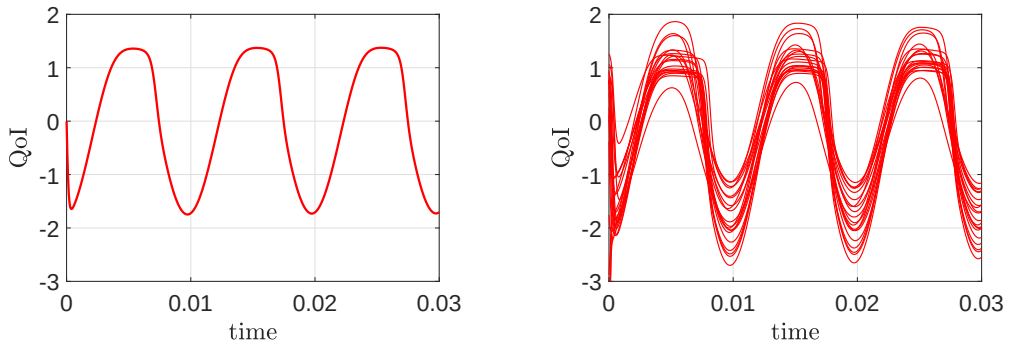


FIG. 3: Trajectory of QoI for mean value of parameters (a) and 20 trajectories of QoI for different parameter samples (b) produced by transistor amplifier circuit

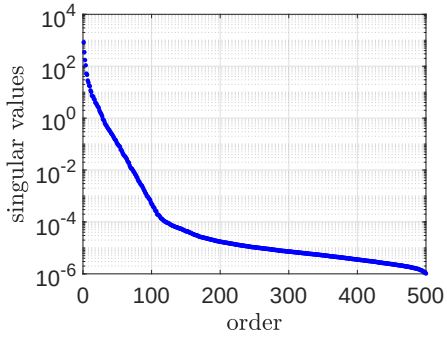


FIG. 4: Singular values from POD for matrix including the samples of the QoI in transistor amplifier example

We train two NNs: a network with two hidden layers including 30 neurons in each layer and a network with three hidden layers including 20 neurons. The activation function used is the hyperbolic tangent relation [Eq. (20)]. The performance function is the mean squared error. The Levenberg-Marquardt method (Du and Swamy, 2014, p. 130) executes the nonlinear optimization. Figure 5 depicts the performance of the training in both NNs. The training is terminated after a maximum number of 1000 iterations in each case, because the stopping criterion based on the validation set is not satisfied yet. Imposing a maximum iteration number represents a kind of regularization in the context of minimization. We observe that the achieved mean squared errors are similar in both NNs. Figure 6 illustrates some samples for the trajectories of the QoI, where the first NN yields the approximations.

Furthermore, we employ the polynomial approximation from Section 2.3 for comparison. Let s_d be the number of basis polynomials up to total degree d depending on 10 variables. We discuss the cases $d = 2, 3, 4$. It follows that $s_2 = 66$, $s_3 = 286$, and $s_4 = 1001$ due to Eq. (11). We use only the training samples [Eq. (21)] in the least squares problem. Hence the validation set becomes just an additional test set. In the case of $d = 4$, the number s_4 of DOFs is larger than the number $k = 1000$ of training samples. Thus we extend the training set by just one sample once. It follows that the polynomial regression changes into a polynomial interpolation in the case of $d = 4$.

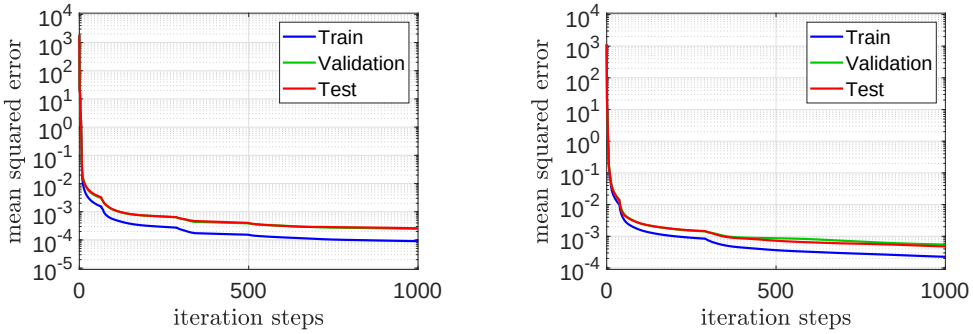


FIG. 5: Performance in training of NNs with two hidden layers (a) and three hidden layers (b) in transistor amplifier example

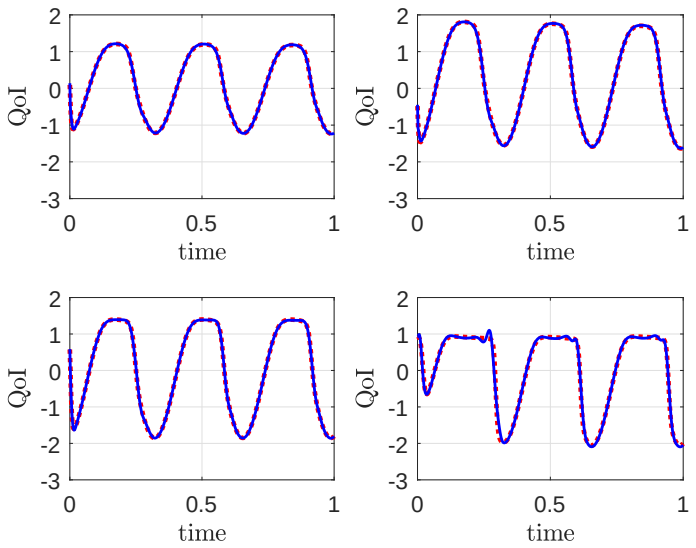


FIG. 6: Trajectories of QoI for four different parameter samples: solution of IVPs (dotted line) and approximation from NN (solid line) in transistor amplifier example (time interval $[0, 0.03]$ is standardized to $[0, 1]$)

Table 1 demonstrates the statistics of the errors [Eq. (24)] based on the discrete $\mathcal{L}^1$-norms. The accuracy of the two NNs coincides. The errors of the polynomial regression with degree 2 are worse. Yet the accuracy of the polynomial approximation improves for increasing total degree in the case of the training set. The polynomial fit of degree 4 produces the same errors as the NNs for the training set, whereas the error of the polynomial approximation is much larger for the validation set as well as the test set. This typical phenomenon is an oversampling with respect to the training samples. In the training routines of the NNs, the consideration of the validation set would stop the training if an overfitting is detected. However, this stopping criterion does not occur in the fitting of our two NNs, because the training terminates after the maximum number of iteration steps. Thus an important observation is that the training of NNs omits overfitting without termination in this example. Moreover, the polynomial interpolation ($d = 4$) features no approximation error in the training set. Hence this mean value is dominated by the approximation error of the reduced basis from the POD approach, which is the error part (2)

TABLE 1: Statistics of errors in approximations by NNs and polynomials in transistor amplifier example

		NN 2 layers	**NN 3 layers**	**Polynomial degree 2**	**Polynomial degree 3**	**Polynomial degree 4**
Mean	Training set	0.0223	0.0223	0.0443	0.0302	0.0223
	Validation set	0.0223	0.0224	0.0465	0.0364	0.6611
	Test set	0.0223	0.0224	0.0462	0.0367	0.6719
Standart deviation	Training set	0.0112	0.0112	0.0217	0.0140	0.0112
	Validation set	0.0099	0.0099	0.0195	0.0165	0.5204
	Test set	0.0109	0.0108	0.0225	0.0194	0.5021

in Section 3.2. We conclude that the approximation errors of the NNs are also negligible, since their mean errors of all sets nearly coincide with the mean error of the polynomial approach for $d = 4$ in the training set.

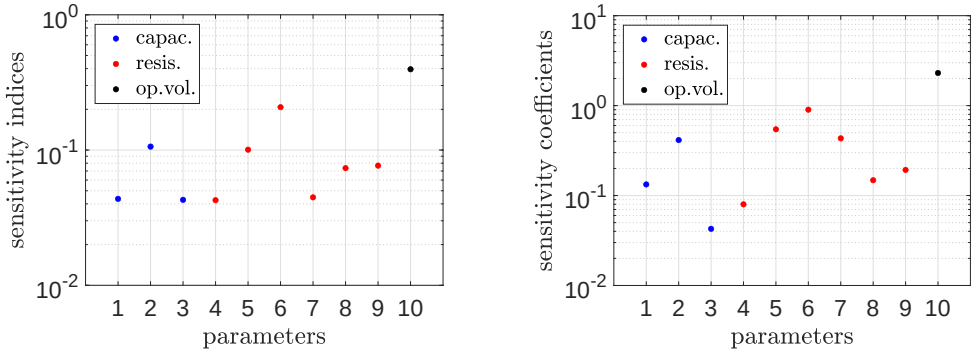
Since the validation set is not used in the polynomial regression, we perform an additional numerical experiment. The polynomials are fitted to the data of the union of training set and validation set (2000 samples). Table 2 shows the statistics of errors. The results are similar to the previous polynomial approach. The effect of overfitting is reduced in the case of degree 4. However, the overfitting is still present and thus the errors are worse for the test set in comparison to the NN models. A polynomial approximation of degree 5 would include 3003 basis polynomials, which is a larger number than the total sample size.

We comment on the computing times. The polynomial regression with data size 1000/1001 required in seconds: 0.05 for degree 2, 0.21 for degree 3, and 0.86 for degree 4. Thus the polynomial approximation is cheap. In contrast, the training of the NN with two layers ran about 41 minutes. There is some potential to reduce the compution work in the training. On the one hand, the number of iterations can be reduced. On the other hand, there are much cheaper iteration techniques in comparison to the Levenberg-Marquardt method. However, the alternative techniques yield worse accuracy in this example.

We compute the total effect sensitivity indices [Eq. (18)] for the varying physical parameters. The approximations $\tilde{a}$ are evaluated on the grid of the Stroud-5 cubature (Stroud, 1971), which is exact for polynomials up to total degree 5. These evaluations yield approximate sensitivities [Eq. (18)] using a non-intrusive method as given in Pulch et al. (2015). Figure 7(a) shows the

TABLE 2: Statistics of errors in approximations by polynomials with joined set (union of training set and validation set) for fitting in transistor amplifier example

		Polynomial degree 2	**Polynomial degree 3**	**Polynomial degree 4**
Mean	Joined set	0.0443	0.0306	0.0249
	Test set	0.0450	0.0334	0.0350
Standart	Joined set	0.0207	0.0142	0.0105
deviation	Test set	0.0223	0.0180	0.0208



(a) (b)

FIG. 7: Sensitivities for different physical parameters (capacitances 1-3, resistances 4-9, operating voltage 10) in transistor amplifier example: (a) total effect sensitivity indices [Eq. (18)] and (b) sensitivity coefficients [Eq. (25)] obtained by NN in semilogarithmic scale

sensitivity indices [Eq. (18)]. We emphasize that these values are computed directly from the mapping [Eq. (8)] without an approximation by NNs or polynomial regression. Alternatively, we consider the trained NN with two hidden layers to obtain the sensitivity coefficients [Eq. (25)] depicted in Fig. 7(b). We recognize a good agreement for the relative positions of the sensitivity indicators in the two concepts. In particular, the ranking of the parameters mostly coincides. The operating voltage exhibits the largest influence. Furthermore, we train an extended NN [Eq. (26)] with three hidden layers of sizes $\hat{N}_1 = 10$ and $\hat{N}_2 = \hat{N}_3 = 30$. Its sensitivity coefficients [Eq. (25)] are shown in Fig. 8. The results are similar to the previous NN.

Finally, we reproduce the total effect sensitivity indices based on the approximate mappings. In Eq. (18), the exact coefficients $\hat{a}_i$ are substituted by the approximations $\tilde{a}_i$. On the one hand, the trained NN with two layers is used, where the NN model is evaluated at the nodes of the Stroud-5 quadrature. On the other hand, the polynomial regression of degree 2, which fits to 1000 samples of the training set, yields the approximation [Eq. (9)], and the coefficients c_i are directly inserted in Eq. (16) to obtain approximations of the partial variances. Figure 9 illustrates the resulting sensitivity indices. We observe that the outcomes of both methods agree roughly.

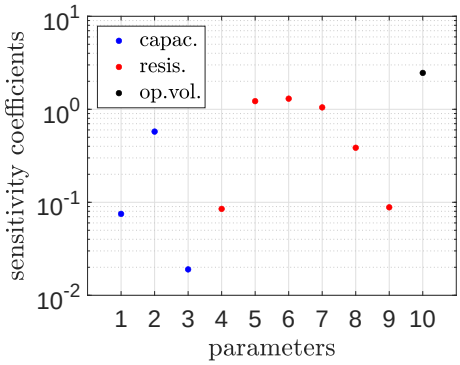


FIG. 8: Sensitivities coefficients [Eq. (25)] for different physical parameters (capacitances 1-3, resistances 4-9, operating voltage 10) using an extended NN in transistor amplifier example

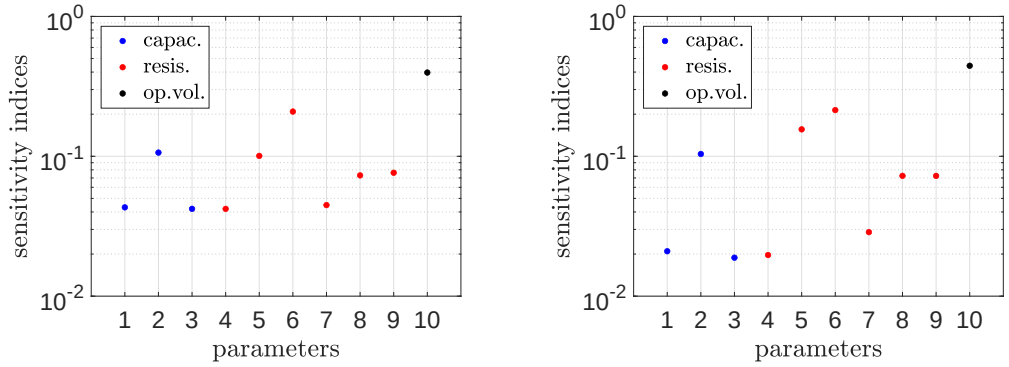


FIG. 9: Total effect sensitivity indices [Eq. (18)] for different physical parameters (capacitances 1-3, resistances 4-9, operating voltage 10) computed using NN (a) and polynomial approximation (b) both in semilogarithmic scale for transistor amplifier

The NN replicates the results in Fig. 7(a). The polynomial regression produces slightly different values for small sensitivity values. This behavior reflects that the approximation of the NN is more accurate.

4.2 Example: Schmitt Trigger

The electric circuit of a Schmitt trigger is illustrated in Fig. 10. There are five resistances, a capacitance, and two bipolar transistors. This circuit acts as an analog-digital converter. A mathematical model is presented in Kampowsky et al. (1992) that consists of five DAEs for five unknown node voltages. The mass matrix and the right-hand side of the system [Eq. (1)] are

$$M = \begin{pmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & C & 0 & -C & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & -C & 0 & C & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix} \qquad f = \begin{pmatrix} -\dfrac{u_1}{R_1} - (1-\gamma)w(u_1 - u_3) \\ -\dfrac{u_2}{R_2} - \dfrac{u_2 - u_4}{R_4} - \gamma w(u_1 - u_3) \\ -w(u_1 - u_3) + \dfrac{u_3}{R_3} - w(u_4 - u_3) \\ -\dfrac{u_4 - u_2}{R_4} - (1-\gamma)w(u_4 - u_3) \\ -\dfrac{u_5}{R_5} - \gamma w(u_4 - u_3) \end{pmatrix} + \begin{pmatrix} \dfrac{u_{\mathrm{in}}}{R_1} \\ \dfrac{u_{\mathrm{op}}}{R_2} \\ 0 \\ 0 \\ \dfrac{u_{\mathrm{op}}}{R_5} \end{pmatrix}.$$

The current-volatage relation of the bipolar transistors is given by Eq. (27) again using the same physical parameters. The differential index of the system is one. We employ a harmonic oscillation [Eq. (28)] with period T and amplitude $A = 5$ as input voltage u_{in}. The QoI is the output voltage $u_{\mathrm{out}} = u_5$.

We arrange a parameter variation in the capacitance, the five resistances, the operating voltage, and the period of the input oscillation. The mean values of the parameters read as capacitance $C = 4 \cdot 10^{-11}$; resistances $R_1 = 200$, $R_2 = 1600$, $R_3 = 100$, $R_4 = 3200$, and $R_5 = 1600$; operating voltage $u_{\mathrm{op}} = 0.2$; and period $T = 0.002$. Ranges of 20% around these mean values are used for each parameter, except for the period varying just 5%. Thus the parameter domain is a cuboid $\Pi \subset \mathbb{R}^8$.

In Eqs. (21)–(23), we incorporate $k = k' = k'' = 500$ samples using a pseudo random number generator. The number of equidistant time points is $m = 1000$ now. Again the NDF schemes yield the numerical solutions of the IVPs within the time interval $[t_0, t_{\mathrm{end}}] = [0, 0.006]$. Starting

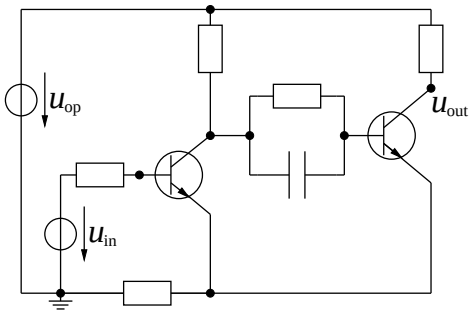


FIG. 10: Electric circuit of a Schmitt trigger

values for the initial values are always zero, whereas a consistent initial value [Eq. (2)] is determined for each parameter tuple by the method. The local error control uses relative tolerance $\varepsilon_{\mathrm{r}} = 10^{-3}$ and absolute tolerance $\varepsilon_{\mathrm{a}} = 10^{-5}$. Figure 11(a) depicts the trajectory associated with the mean value of the parameters. Although the solution of the DAE system is continuous, the output voltage exhibits fast transitions, which behave like jumps. The sinusoidal input signal [Eq. (28)] is transformed into a digital output signal. Figure 11(b) illustrates the trajectories for several parameter samples. The variation of the period shifts the locations of the jumps.

We perform the POD approach for the training samples. Figure 12 displays the singular values of the decomposition [Eq. (4)]. The decay of the singular values is slower in comparison to the previous example shown in Fig. 4, since the trajectories vary to a higher extent. We set the reduced dimension to $r = 11$, which is the smallest number satisfying Eq. (7) with the threshold $\delta = 0.99$.

We train two NNs of the same sizes as in the example of Section 4.1. A conjugate-gradient method (Du and Swamy, 2014, p. 136) solves the nonlinear optimization problem iteratively. On the one hand, a single iteration step is much cheaper in comparison to the Levenberg-Marquardt method. On the other hand, more iteration steps are required in total. Yet the total computation work is significantly lower now. Figure 13 shows the training procedure of the two NNs. The iteration is terminated at the 5189th step and the 10,258th step, respectively, since the performance function does not decrease any more on the validation set in both cases. In the training, the computing times were 22.2 seconds and 43.8 seconds, respectively.

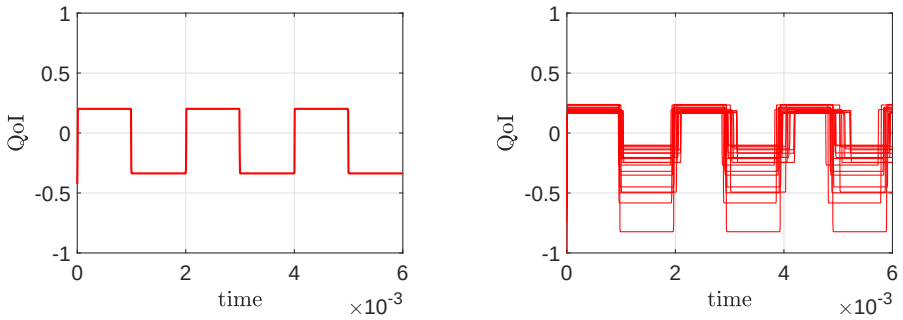


FIG. 11: Trajectory of QoI for mean value of parameters (a) and twenty trajectories of QoI for different parameter samples (b) produced by Schmitt trigger circuit

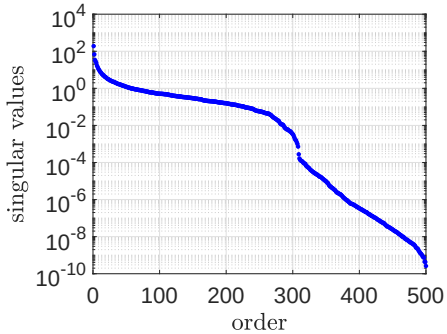


FIG. 12: Singular values from POD for matrix including the samples of the QoI in Schmitt trigger example

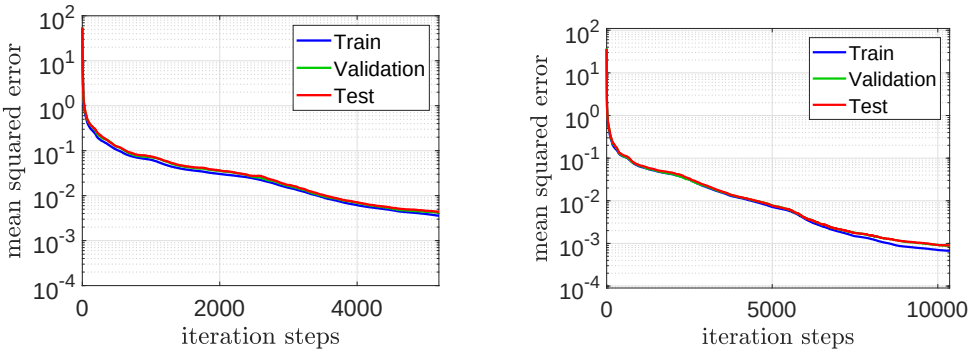


FIG. 13: Performance in training of NNs with two hidden layers (a) and three hidden layers (b) in Schmitt trigger example

We use the first NN to approximate the trajectories of the QoI. Figure 14 demonstrates the approximations together with the original trajectories of the time integration. We observe a good agreement of the amplitudes and a good localization of the jumps. However, incorrect oscillations occur close to the jumps, which are caused by the reduced basis in the POD approximation.

We check the NNs against the polynomial regression from Section 2.3. The approximations of total degree $d = 2, 3, 4$ are computed, where the number of basis polynomials is $s_2 = 45$, $s_3 = 165$, and $s_4 = 495$, respectively. Hence we solve linear least squares problems using the training set [Eq. (21)]. The case of $d = 4$ nearly coincides with a polynomial interpolation of the training set due to $s_4 \approx k$. Table 3 contains the statistics of the discrete $\mathcal{L}^1$-errors [Eq. (24)]. We observe the same behavior as in the example of Section 4.1. Again the NNs are better than a straightforward polynomial approximation.

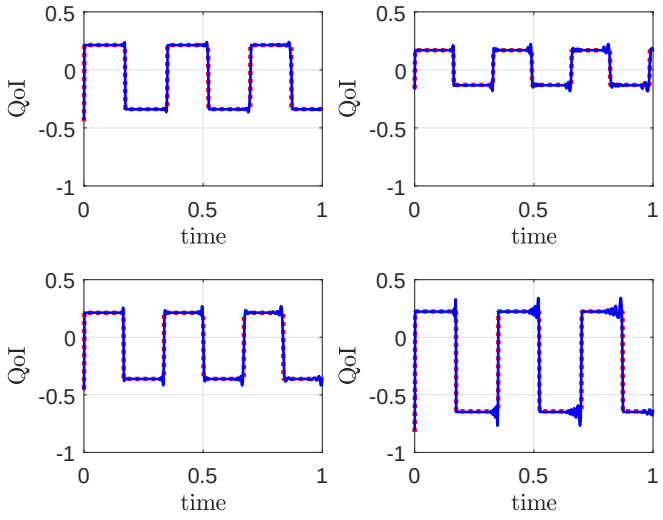


FIG. 14: Trajectories of QoI for four different parameter samples: solution of IVPs (dotted line) and approximation from NN (solid line) in Schmitt trigger example (time interval $[0, 0.006]$ is standardized to $[0, 1]$)

TABLE 3: Statistics of errors in approximations by NNs and polynomials in Schmitt trigger example

		NN 2 layers	NN 3 layers	Polynomial degree 2	Polynomial degree 3	Polynomial degree 4
Mean	Training set	0.0089	0.0089	0.0246	0.0180	0.0089
	Validation set	0.0092	0.0090	0.0269	0.0254	0.1896
	Test set	0.0093	0.0092	0.0276	0.0263	0.1792
Standart deviation	Training set	0.0029	0.0030	0.0087	0.0060	0.0030
	Validation set	0.0034	0.0033	0.0093	0.0105	0.1463
	Test set	0.0034	0.0035	0.0095	0.0107	0.1190

Again, we also fit the polynomials to the data of the union of training set and validation set (1000 samples). Table 4 depicts the statistical errors. An overfitting occurs again and thus the approximation of the NNs is more accurate. A polynomial approximation of degree 5 would include 1287 basis polynomials, where the number of degrees of freedom is larger than the size of the joined sample set.

Finally, we compute the total effect sensitivity indices [Eq. (18)] of the exact mapping as well as the sensitivity coefficients [Eq. (25)] of the first NN shown in Fig. 15. The variance-based concept identifies the operating voltage as most important, whereas the period dominates in the weight-based approach. Although the varying period changes the positions of the jumps, the (Euclidean) norm of the basis coefficients in Eq. (18) remains nearly the same. The total effect sensitivity indices of the first two parameters are tiny. A more detailed investigation confirms that these two parameters hardly influence the QoI, while they have some impact on other variables of the solution. However, the sensitivity coefficients [Eq. (25)] from the NN are not small for the two parameters. Thus we construct an extended NN [Eq. (26)] with three hidden layers of sizes $\hat{N}_1 = 8$ and $\hat{N}_2 = \hat{N}_3 = 30$. Figure 16 displays the computed sensitivity coefficients [Eq. (25)]. Now the first two parameters are correctly detected as insignificant variables with respect to the observed QoI.

5. CONCLUSIONS

We approximated trajectories of a QoI, which is the output of a nonlinear dynamical system. Artificial NNs were fitted to data obtained by a POD. The numerical computations of test examples demonstrate that neural networks with relatively low numbers of hidden layers are already sufficiently accurate. Moreover, the NNs are superior in comparison to a multivariate polynomial approximation by regression. In addition, a variance-based sensitivity analysis of the input–output

TABLE 4: Statistics of errors in approximations by polynomials with joined set (union of training set and validation set) for fitting in Schmitt trigger example

		Polynomial degree 2	Polynomial degree 3	Polynomial degree 4
Mean	Joined set	0.0250	0.0192	0.0140
	Test set	0.0268	0.0228	0.0258
Standart deviation	Joined set	0.0087	0.0067	0.0046
	Test set	0.0091	0.0084	0.0124

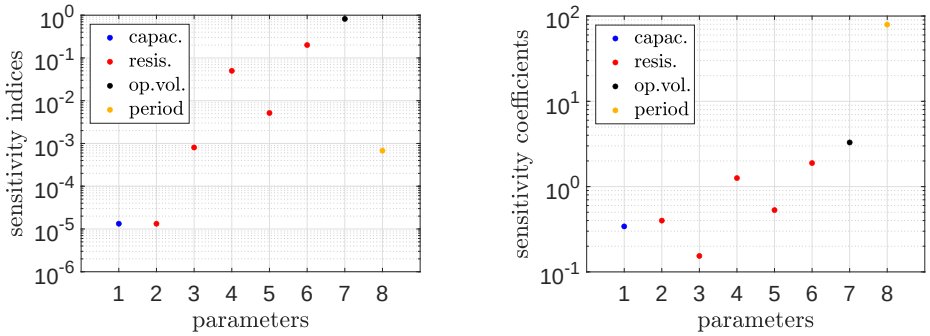


FIG. 15: Sensitivities for different physical parameters (capacitance 1, resistances 2-6, operating voltage 7, period 8) in Schmitt trigger example: (a) total effect sensitivity indices [Eq. (18)] and (b) sensitivity coefficients [Eq. (25)] obtained by NN in semilogarithmic scale

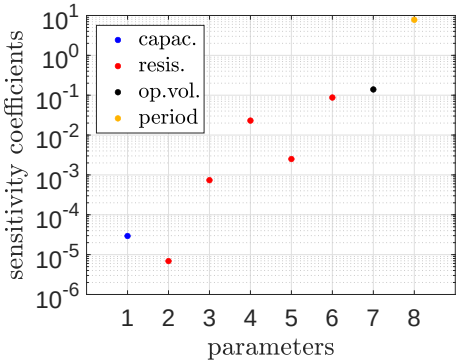


FIG. 16: Sensitivities coefficients [Eq. (25)] for different physical parameters (capacitance 1, resistances 2-6, operating voltage 7, period 8) using an extended NN in Schmitt trigger example

mapping was considered. We introduced an alternative concept based on the weights in a trained NN. The variance-based technique already becomes cheap when an NN is applied to approximate the mapping. Alternatively, the sensitivity concept using the weights does not significantly save computational effort but it provides some insight into the input–output relation of an NN.

REFERENCES

Antoulas, A.C., *Approximation of Large-Scale Dynamical Systems*, Philadelphia: SIAM, 2005.

Benner, P., Gugercin, S., and Willcox, K., A Survey of Projection-Based Model Reduction Methods for Parametric Dynamical Systems, *SIAM Review*, vol. **57**, no. 4, pp. 483–531, 2015.

Du, K. and Swamy, M., *Neural Networks and Statistical Learning*, Berlin: Springer, 2014.

Genzel, M. and Kutyniok, G., Artificial Neural Networks, *GAMM Rundbrief*, vol. **2**, pp. 12–18, 2019.

Golub, G. and van Loan, C., *Matrix Computations*, Baltimore: Johns Hopkins University Press, 1996.

Goodfellow, I., Bengio, Y., and Courville, A., *Deep Learning*, Cambridge, MA: MIT Press, 2017.

Hairer, E. and Wanner, G., *Solving Ordinary Differential Equations. Vol. 2: Stiff and Differential-Algebraic Equations*, Berlin: Springer, 1996.

Hesthaven, J. and Ubbiali, S., Non-Intrusive Reduced Order Modeling of Nonlinear Problems Using Neural Networks, *J. Comput. Phys.*, vol. **363**, pp. 55–78, 2018.

Kampowsky, W., Rentrop, P., and Schmidt, W., Classification and Numerical Solution of Electric Circuits, *Surv. Math. Ind.*, vol. **2**, pp. 23–65, 1992.

Kunisch, K. and Volkwein, S., Galerkin Proper Orthogonal Decomposition Methods for Parabolic Problems, *Numer. Math.*, vol. **90**, pp. 117–148, 2001.

Mifsud, M., MacManus, D., and Shaw, S., A Variable-Fidelity Aerodynamic Model Using Proper Orthogonal Decomposition, *J. Numer. Meth. Fluids*, vol. **82**, pp. 646–663, 2016.

Montavon, G., Samek, W., and Muller, K., Methods for Interpreting and Understanding Deep Neural Networks, *Digital Signal Process.*, vol. **73**, pp. 1–15, 2018.

Pulch, R., Model Order Reduction for Random Nonlinear Dynamical Systems and Low-Dimensional Representations for Their Quantities of Interest, *Math. Comput. Simulat.*, vol. **166**, pp. 76–92, 2019.

Pulch, R. and Narayan, A., Sensitivity Analysis of Random Linear Dynamical Systems Using Quadratic Outputs, *J. Comput. Appl. Math.*, 2019. DOI: 10.1016/j.cam.2019.112491

Pulch, R., ter Maten, J., and Augustin, F., Sensitivity Analysis and Model Order Reduction for Random Linear Dynamical Systems, *Math. Comput. Simulat.*, vol. **111**, pp. 80–95, 2015.

Qin, T., Wu, K., and Xiu, D., Data Driven Governing Equations Approximation Using Deep Neural Networks, *J. Comput. Phys.*, vol. **395**, pp. 620–635, 2019.

Saltelli, A., Ratto, M., Andres, T., Campolongo, F., Cariboni, J., Gatelli, D., Saisana, M., and Tarantola, S., *Global Sensitivity Analysis*, John Wiley and Sons, Ltd., 2008.

Seber, G. and Lee, A., *Linear Regression Analysis*, Hoboken, NJ: John Wiley and Sons, Inc., 2003.

Sobol, I., Global Sensitivity Indices for Nonlinear Mathematical Models and Their Monte Carlo Estimates, *Math. Comput. Simulat.*, vol. **55**, pp. 271–280, 2001.

Sobol, I. and Kucherenko, S., Derivative based Global Sensitivity Measures and Their Link with Global Sensitivity Indices, *Math. Comput. Simulat.*, vol. **79**, pp. 3009–3017, 2009.

Stoer, J. and Bulirsch, R., *Introduction to Numerical Analysis*, Berlin: Springer, 2002.

Stroud, A., *Approximate Calculation of Multiple Integrals*, Upper Saddle River, NJ: Prentice-Hall, 1971.

Sudret, B., Global Sensitivity Analysis Using Polynomial Chaos Expansions, *Reliability Eng. Syst. Safety*, vol. **93**, no. 7, pp. 964–979, 2008.

Wang, X., The Effect of Regularization Coefficient on Polynomial Regression, *J. Phys.: Conf. Ser.*, vol. **1213**, p. 042054, 2019.

Xiu, D., *Numerical Methods for Stochastic Computations: A Spectral Method Approach*, Princeton, NJ: Princeton University Press, 2010.

Yu, J. and Hesthaven, J., Flowfield Reconstruction Method Using Artificial Neural Network, *AIAA J.*, vol. **57**, 2019.