Theory, Algorithms, and Software for Physics-Informed Deep Learning

by

Lu Lu

B.Eng., Tsinghua University; Beijing, China, 2013

B.Ec., Tsinghua University; Beijing, China, 2013

Minor, Tsinghua University; Beijing, China, 2013

M.Sc., Brown University; Providence, RI, USA, 2016

M.Sc., Brown University; Providence, RI, USA, 2017

M.Sc., Brown University; Providence, RI, USA, 2019

A dissertation submitted in partial fulfillment of the

requirements for the degree of Doctor of Philosophy

in the Division of Applied Mathematics at Brown University

PROVIDENCE, RHODE ISLAND

May 2020

This dissertation by Lu Lu is accepted in its present form
by the Division of Applied Mathematics as satisfying the
dissertation requirement for the degree of Doctor of Philosophy.

Date_____ _____

George Em Karniadakis, Ph.D., Advisor

Recommended to the Graduate Council

Date_____ _____

Matthew T. Harrison, Ph.D., Reader

Date_____ _____

Panos Stinis, Ph.D., Reader

Approved by the Graduate Council

Date_____ _____

Andrew G. Campbell, Dean of the Graduate School

Vitae

Lu Lu finished his Bachelor of Engineering degree in Energy, Power System and Automation, Bachelor of Economics degree in Economics, and Minor in Computer Technology and Application, in July 2013 at Tsinghua University in Beijing, China. He then moved to Providence, Rhode Island, in pursuit of a Doctor of Philosophy degree in Applied Mathematics at Brown University, where he also received a Master of Science degree in Engineering in 2016, a Master of Science degree in Applied Mathematics in 2017, and a Master of Science degree in Computer Science in 2019.

Lu's research interest lies at the interface of applied mathematics, physics, computational biology, and computer science. Advised by Professor George Em Karniadakis, during his Ph.D. study, Lu has completed eighteen publications, including one in *Science Advances* and two in *Proceedings of the National Academy of Sciences*, and was invited to present his work in nearly twenty conferences and seminars, including the SIAM Annual Meeting (2017), the Conference on Neural Information Processing Systems (NeurIPS) Workshop (2019), the Joint Mathematics Meetings (2020), and the Association for the Advancement of Artificial Intelligence (AAAI) Spring Symposium (2020).

Lu has been a teaching assistant in four courses, including Advanced Fluid Mechanics (spring 2015), Computational Probability and Statistics (fall 2017), Op-

erations Research: Probability Models (spring 2018), and Numerical Solution of Partial Differential Equations I (fall 2018).

In addition, Lu has actively participated in professional service activities. He has organized a minisymposium on Machine Learning for Physical Systems at the SIAM Conference on Mathematics of Data Science (2020). He has also served as a peer reviewer for nine scientific journals and conferences, including IEEE Transactions on Neural Networks and Learning Systems, Journal of Computational Physics, NeurIPS (2019), and International Conference on Machine Learning (2020), and has reviewed about twenty manuscripts.

Publications

*Contributed equally

Journal papers

1. Y. Chen, **L. Lu**, G. E. Karniadakis, & L. D. Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *Optics Express*, 28(8), 11618–11633, 2020.

2. **L. Lu***, M. Dao*, P. Kumar, U. Ramamurty, G. E. Karniadakis, & S. Suresh. Extraction of mechanical properties of materials through deep learning from instrumented indentation. *Proceedings of the National Academy of Sciences*, 117(13), 7052–7062, 2020.

3. G. Pang*, **L. Lu***, & G. E. Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4), A2603–A2626, 2019.

4. **L. Lu***, Z. Li*, H. Li*, X. Li, P. G. Vekilov, & G. E. Karniadakis. Quantitative prediction of erythrocyte sickling for the development of advanced sickle cell therapies. *Science Advances*, 5(8), eaax3905, 2019. (**Highlighted on *Science Advances* homepage**)

5. D. Zhang, **L. Lu**, L. Guo, & G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397, 108850, 2019.

6. H. Li*, **L. Lu***, X. Li, P. A. Buffet, M. Dao, G. E. Karniadakis, & S. Suresh. Mechanics of diseased red blood cells in human spleen and consequences for hereditary blood disorders. *Proceedings of the National Academy of Sciences*, 115(38), 9574–9579, 2018.

7. H. Li, D. Papageorgiou, H. Y. Chang, **L. Lu**, J. Yang, & Y. Deng. Synergistic integration of laboratory and numerical approaches in studies of the biomechanics of diseased red blood cells. *Biosensors*, 8(3), 76, 2018.

8. **L. Lu***, Y. Deng*, X. Li, H. Li, & G. E. Karniadakis. Understanding the twisted structure of amyloid fibrils via molecular simulations. *The Journal of Physical Chemistry B*, 122(49), 11302–11310, 2018.

9. H. Li, J. Yang, T. T. Chu, R. Naidu, **L. Lu**, R. Chandramohanadas, M. Dao & G. E. Karniadakis. Cytoskeleton remodeling induces membrane stiffness and stability changes of maturing reticulocytes. *Biophysical Journal*, 114(8), 2014–2023, 2018. (**Highlighted on *Biophysical Journal* homepage**)

10. H. Li, H. Y. Chang, J. Yang, **L. Lu**, Y. H. Tang, & G. Lykotrafitis. Modeling biomembranes and red blood cells by coarse-grained particle methods. *Applied Mathematics and Mechanics*, 39(1), 3–20, 2018.

11. **L. Lu**, H. Li, X. Bian, X. Li, & G. E. Karniadakis. Mesoscopic adaptive resolution scheme toward understanding of interactions between sickle cell fibers. *Biophysical Journal*, 113(1), 48–59, 2017. (**Cover Article**)

12. Y. H. Tang*, **L. Lu***, H. Li, C. Evangelinos, L. Grinberg, V. Sachdeva, & G. E. Karniadakis. OpenRBC: A fast simulator of red blood cells at protein

resolution. *Biophysical Journal*, 112(10), 2030–2037, 2017. (**Highlighted on *Biophysical Journal* homepage**)

13. **L. Lu**, X. Li, P. G. Vekilov, & G. E. Karniadakis. Probing the twisted structure of sickle hemoglobin fibers via particle simulations. *Biophysical Journal*, 110(9), 2085–2093, 2016. (**Highlighted on *Biophysical Journal* homepage**)

14. **L. Lu**, X. Zhang, Y. Yan, J. M. Li, & X. Zhao. Theoretical analysis of natural-gas leakage in urban medium-pressure pipelines. *Journal of Environment and Human*, 1(2), 71–86, 2014.

Preprints

1. **L. Lu**, P. Jin, & G. E. Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

2. **L. Lu**, X. Meng, Z. Mao, & G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

3. P. Jin[*], **L. Lu**[*], Y. Tang, & G. E. Karniadakis. Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness. *arXiv preprint arXiv:1905.11427*, 2019.

4. **L. Lu**[*], Y. Shin[*], Y. Su, & G. E. Karniadakis. Dying ReLU and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

5. **L. Lu**, Y. Su, & G. E. Karniadakis. Collapse of deep and narrow neural nets. *arXiv preprint arXiv:1808.04947*, 2018.

Workshop Papers

1. **L. Lu**, X. Meng, Z. Mao, & G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *AAAI Spring Symposium on Combining Artificial Intelligence and Machine Learning with Physical Sciences*, 2020.

2. **L. Lu**, X. Meng, Z. Mao, & G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *Conference on Neural Information Processing Systems Workshop on Machine Learning and the Physical Sciences*, 2019.

Patents

1. S. Suresh, **L. Lu**, G. E. Karniadakis, & M. Dao. Machine learning techniques for estimating mechanical properties of solid materials. *US Provisional Patent* filed on June 24, 2019.

2. X. Dong, J. M. Li, Y. Yan, H. Zhang, **L. Lu**, J. Wang, & H. Xiao. A test device and method for simulating natural gas leakage in soil. *China Invention Patent* CN103712755A, filed on June 14, 2013, and issued on April 9, 2014.

Acknowledgments

First and foremost, I would like to thank my advisor, Professor George Em Karniadakis, who provides me with the opportunity to study and conduct research in Crunch Group at Brown University. He is a great mentor, and was always available to provide me with visionary guidance throughout the entire course of my Ph.D. study. I am so grateful for him for all his support in the past six years, and I could never ask for a better mentor.

I would also like to express my gratitude to the other members in my dissertation committee: Professor Matthew T. Harrison and Dr. Panos Stinis for sparing time reading through my dissertation, and providing valuable feedback and suggestions.

Special thanks to my collaborators. Dr. Xuejin Li and Professor Peter Vekilov helped me to finish my first project on sickle cell disease. Professor Subra Suresh, Dr. Ming Dao, Dr. He Li, Yixiang Deng, and Dr. Yu-Hang Tang have collaborated closely with me on many projects. I have learned a great deal from Dr. Xin Bian and Dr. Zhen Li on dissipative particle dynamics and smoothed-particle hydrodynamics. Dr. Guofei Pang and Dr. Dongkun Zhang introduced to me the fractional and stochastic partial differential equations, respectively. Dr. Yeonjong Shin and Dr. Yanhui Su helped me finish my first theoretical work on deep learning. I would like to thank other collaborators, including Pengzhan Jin, Dr. Xuhui Meng,

Dr. Zhiping Mao, Yuyao Chen, Professor Luca Dal Negro, Professor Ling Guo, and Dr. Leopold Grinberg.

My thanks also go to all our group's current and former colleagues, including Ansel Blumers, Shengze Cai, Hung-Yu Chang, Sheng Chen, Xiaoli Chen, Mingge Deng, Shijing Cheng, Ehsan Kharazmi, Changho Kim, Seungjoon Lee, Chensen Lin, Guang Lin, Qin Lou, Pavan Mehta, Paris Perdikaris, Jinlian Ren, Fangying Song, Nathaniel Task, Yi Wang, Zhicheng Wang, Mengjia Xu, Liu Yang, Xiu Yang, Minglang Yin, Fanhai Zeng, Enrui Zhang, Zhen Zhang, Zhongqiang Zhang, Lifei Zhao, Xiaoning Zheng, and Zongren Zou for the wonderful time that we have enjoyed together.

Further, I would like to thank the Open Graduate Education Program at Brown University, which provides me the opportunity and support to pursue a master's degree in Computer Science. The knowledge in Computer Science enables me to write this dissertation and forge my career pathway.

Last but not least, I have my utmost gratitude towards my mother Guifeng Zhu and my father Zhengfa Lu. Without your endless love, support and encouragement, I would not be able to complete the journey of this Ph.D. study.

Contents

List of Tables

List of Figures

Chapter

ONE

Introduction

One of the most amazing developments across scientific disciplines in the last three centuries has been the genesis of mathematical models that can be employed to predict and interpret physical and biological phenomena related to fundamental laws of classical physics and beyond. From Archimedes's law, to Newton's law, to the Navier-Stokes equations, to the relativity theories, or the uncertainty principle, the description of linear and nonlinear systems has followed the mathematics prevalent at the time or prior of these developments. While such models have served science well, new fundamental developments are slow and have not established any governing principles in some fields, e.g., in the emerging field of social dynamics. In the era of complexity and big data, the question is then if there are any alternative, simpler, and more compact representations that can be employed to further promote scientific inquiry in established fields where unsolved problems remain, e.g., turbulence in classical physics, or discover new laws in new fields, e.g., in systems dominated by social and behavioral interactions.

Recent developments of deep learning (DL), i.e., deep neural networks (NNs), provide us with opportunities that cannot be tackled solely through traditional methods for scientific applications. In addition to well-known applications of machine learning (ML) algorithms in such fields as image/video analysis [106, 200] and natural language processing [228], ML has also been used for various engineering problems, such as in the discovery of new materials [190] and in healthcare [63]. However, DL usually requires a large amount of data to train the neural network model, and in many engineering problems, it is often difficult to obtain the necessary data of high accuracy.

To address this challenge, there is a significant opportunity for DL to complement and also augment significantly traditional scientific domain knowledge, including physical principles, constraints, symmetries, conservation laws, com-

putational simulations, etc. By incorporating such scientific domain knowledge in DL, it is expected to improve accuracy, interpretability, and robustness of DL models, while simultaneously reducing data requirements and accelerating DL model training and prediction. Progress in this research direction requires the answer to the following question, which we will address in Chapters 4, 5 and 6:

- How should domain knowledge be incorporated into deep learning?

Building the mathematical, statistical, and information-theoretic foundations of deep learning is especially vital to establishing assurance and thus realizing the potential of DL for science and engineering, where DL models would otherwise fail to be adopted. However, despite the remarkable success in the last 15 years, our theoretical understanding of deep learning is lagging behind, which is currently a serious bottleneck of DL for scientific discovery. We will address the following question of assurance and also provide theoretical insights to guide the practical applications in Chapters 2 and 3:

- Whether a deep learning model is appropriately trained and used for the task for which it is intended, including whether prediction errors can be meaningfully bounded?

In what follows, we provide a summary of how the content of this dissertation has been organized.

Part I: Theoretical analysis of deep learning

Dying ReLU. The dying ReLU refers to the problem when ReLU neurons become inactive and only output 0 for any input. There are many empirical and heuristic explanations of why ReLU neurons die. However, little is known about its theoretical analysis. In Chapter 2, we rigorously prove that a deep ReLU network will eventually die in probability as the depth goes to infinite. Several methods have been proposed to alleviate the dying ReLU. Perhaps, one of the simplest treatments is to modify the initialization procedure. One common way of initializing weights and biases uses symmetric probability distributions, which suffers from the dying ReLU. We thus propose a new initialization procedure, namely, a randomized asymmetric initialization. We show that the new initialization can effectively prevent the dying ReLU. All parameters required for the new initialization are theoretically designed. Numerical examples are provided to demonstrate the effectiveness of the new initialization procedure.

Generalization. The accuracy of deep learning, i.e., deep neural networks, can be characterized by dividing the total error into three main types: approximation error, optimization error, and generalization error. Whereas there are some satisfactory answers to the problems of approximation and optimization, much less is known about the theory of generalization. Most existing theoretical works for generalization fail to explain the performance of neural networks in practice. To derive a meaningful bound, In Chapter 3 we study the generalization error of neural networks for classification problems in terms of data distribution and neural network smoothness. We introduce the *cover complexity* (CC) to measure the difficulty of learning a data set and the *inverse of the modulus of continuity* to

quantify neural network smoothness. A quantitative bound for expected accuracy/error is derived by considering both the CC and neural network smoothness. We validate our theoretical results by several data sets of images. The numerical results confirm that the expected error of trained networks scaled with the square root of the number of classes has a linear relationship with respect to the CC. We also observe a clear consistency between test loss and neural network smoothness during the training process. In addition, we show that neural network smoothness decreases when the network size increases, while the smoothness is insensitive to training dataset size.

Part II: Algorithms for physics-informed deep learning

Multi-fidelity neural networks. Instrumented indentation has been developed and widely utilized as one of the most versatile and practical means of extracting mechanical properties of materials. This method is particularly desirable for those applications where it is difficult to experimentally determine the mechanical properties using stress–strain data obtained from coupon specimens. Such applications include material processing and manufacturing of small and large engineering components and structures involving the following: three-dimensional (3D) printing, thin-film and multilayered structures, and integrated manufacturing of materials for coupled mechanical and functional properties. Here, in Chapter 4 we utilize the latest developments in neural networks, including a multifidelity approach whereby deep-learning algorithms are trained to extract elastoplastic properties of metals and alloys from instrumented indentation results using multiple datasets for desired levels of improved accuracy. We have established algorithms for solving inverse problems by recourse to single, dual, and multi-

ple indentation and demonstrate that these algorithms significantly outperform traditional brute force computations and function-fitting methods. Moreover, we present several multifidelity approaches specifically for solving the inverse indentation problem which 1) significantly reduce the number of high-fidelity datasets required to achieve a given level of accuracy, 2) utilize known physical and scaling laws to improve training efficiency and accuracy, and 3) integrate simulation and experimental data for training disparate datasets to learn and minimize systematic errors. The predictive capabilities and advantages of these multifidelity methods have been assessed by direct comparisons with experimental results for indentation for different commercial alloys, including two wrought aluminum alloys and several 3D printed titanium alloys.

Physics-informed neural networks. Deep learning has achieved remarkable success in diverse applications; however, its use in solving partial differential equations (PDEs) has emerged only recently. In Chapter 5 we present an overview of physics-informed neural networks (PINNs), which embed a PDE into the loss of the neural network using automatic differentiation. The PINN algorithm is simple, and it can be applied to different types of PDEs, including integro-differential equations, fractional PDEs, and stochastic PDEs. Moreover, from the implementation point of view, PINNs solve inverse problems as easily as forward problems. We propose a new residual-based adaptive refinement (RAR) method to improve the training efficiency of PINNs. For pedagogical reasons, we compare the PINN algorithm to a standard finite element method.

DeepONet: Learning nonlinear operators. It is widely known that neural networks (NNs) are universal approximators of continuous functions, however, a less

known but powerful result is that a NN with a single hidden layer can approximate accurately any nonlinear continuous operator. This universal approximation theorem of operators is suggestive of the potential of NNs in learning from scattered data any continuous operator or complex system. To realize this theorem, in Chapter 6 we design a new NN with small generalization error, the deep operator network (DeepONet), consisting of a NN for encoding the discrete input function space (branch net) and another NN for encoding the domain of the output functions (trunk net). We demonstrate that DeepONet can learn various explicit operators, e.g., integrals and fractional Laplacians, as well as implicit operators that represent deterministic and stochastic differential equations. We study, in particular, different formulations of the input function space and its effect on the generalization error.

Part III: Open-source software

In Chapter 5, we also present a Python library for PINNs, DeepXDE, which is designed to serve both as an education tool to be used in the classroom as well as a research tool for solving problems in computational science and engineering. Specifically, DeepXDE can solve forward problems given initial and boundary conditions, as well as inverse problems given some extra measurements. DeepXDE supports complex-geometry domains based on the technique of constructive solid geometry, and enables the user code to be compact, resembling closely the mathematical formulation. We introduce the usage of DeepXDE and its customizability, and we also demonstrate the capability of PINNs and the user-friendliness of DeepXDE for five different examples. More broadly, DeepXDE contributes to the more rapid development of the emerging Scientific Machine Learning field.

Chapter

TWO

Dying ReLU and Initialization: Theory and Numerical Examples

Manuscript information

A version of this chapter appeared in [137]: L. Lu*, Y. Shin*, Y. Su, & G. E. Karniadakis. Dying ReLU and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

Author contributions: L.L. conceived the dying ReLU problem and performed the experiments. L.L., Y.S., and Y.S. developed the theory. Y.S. developed the randomized asymmetric initialization. L.L., Y.S., Y.S., and G.E.K. wrote the manuscript. G.E.K. supervised the project.

2.1 Introduction

The rectified linear unit (ReLU), $\max\{x, 0\}$, is one of the most successful and widely-used activation functions in deep learning [123, 183, 154]. The success of ReLU is based on its superior training performance [72, 204] over other activation functions such as the logistic sigmoid and the hyperbolic tangent [71, 125]. The ReLU has been used in various applications including image classification [115, 209], natural language processes [140], speech recognition [90], and game intelligence [195], to name a few.

The use of gradient-based optimization is inevitable in training deep neural networks. It has been widely known that the deeper a neural network is, the harder it is to train [203, 55]. A fundamental difficulty in training deep neural networks is the vanishing and exploding gradient problem [177, 81, 34]. The dying ReLU is a

kind of vanishing gradient, which refers to a problem when ReLU neurons become inactive and only output 0 for any input. It has been known as one of the obstacles in training deep ReLU neural networks [215, 2]. To overcome this problem, a number of methods have been proposed. Broadly speaking, these can be categorized into three general approaches. One approach modifies the network architectures. This includes but not limited to the changes in the number of layers, the number of neurons, network connections, and activation functions. In particular, many activation functions have been proposed to replace the ReLU [140, 86, 48, 113]. However, the performance of other activation functions varies on different tasks and data sets [183] and it typically requires a parameter to be turned. Thus, the ReLU remains one of the popular activation functions due to its simplicity and reliability. Another approach introduces additional training steps. This includes several normalization techniques [95, 189, 217, 10, 225] and dropout [202]. One of the most successful normalization techniques is the batch normalization [95]. It is a technique that inserts layers into the deep neural network that transform the output for the batch to be zero mean unit variance. However, batch normalization increases by 30% the computational overhead to each iteration [149]. The third approach modifies only weights and biases initialization procedure without changing any network architectures or introducing additional training steps [125, 71, 86, 192, 149]. The third approach is the topic of our work presented in this chapter.

The intriguing ability of gradient-based optimization is perhaps one of the major contributors to the success of deep learning. Training deep neural networks using gradient-based optimization fall into the noncovex nonsmooth optimization. Since a gradient-based method is either a first- or a second-order method, and once converged, the optimizer is either a local minimum or a saddle point. The authors

of [67] proved that the existence of local minima poses a serious problem in training neural networks. Many researchers have been putting immense efforts to mathematically understand the gradient method and its ability to solve nonconvex nonsmooth problems. Under various assumptions, especially on the landscape, many results claim that the gradient method can find a global minimum, can escape saddle points, and can avoid spurious local minima [126, 6, 68, 69, 242, 224, 229, 54, 56, 55, 100]. However, these assumptions do not always hold and are provably false for deep neural networks [187, 107, 7]. This further limits our understanding on what contributes to the success of the deep neural networks. Often, theoretical conditions are impossible to be met in practice.

Where to start the optimization process plays a critical role in training and has a significant effect on the trained result [158]. This chapter focuses on a particular kind of bad local minima due to a bad initialization. Such a bad local minimum causes the dying ReLU. Specifically, we consider the worst case of dying ReLU, where the entire network dies, i.e., the network becomes a constant function. We refer this as *the dying ReLU neural networks* (NNs). This phenomenon could be well illustrated by a simple example. Suppose $f(x) = |x|$ is a target function we want to approximate using a ReLU network. Since $|x| = \mathrm{ReLU}(x) + \mathrm{ReLU}(-x)$, a 2-layer ReLU network of width 2 can exactly represent $|x|$. However, when we train a deep ReLU network, we frequently observe that the network is collapsed. This trained result is shown in Fig. 2.1. Our 1,000 independent simulations show that there is a high probability (more than 90%) for the deep ReLU network to collapse to a constant function. In this example, we employ a 10-layer ReLU network of width 2 which should perfectly recover $f(x) = |x|$.

Almost all common initialization schemes in training deep neural networks use symmetric probability distributions around 0. For example, zero mean uniform

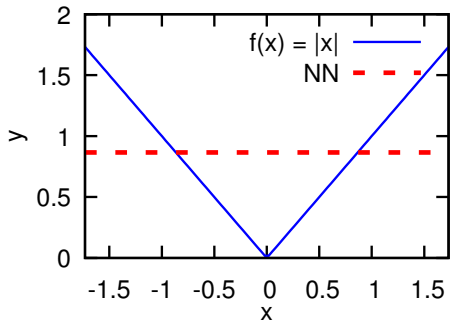


Figure 2.1: An approximation result for $f(x) = |x|$ using a 10-layer ReLU neural network of width 2. Among 1,000 independent simulations, this trained result is obtained with more than 90% probability. One of the most popular initialization procedures [86] is employed.

distributions and zero mean normal distributions were proposed and used in [125, 71, 86]. We show that when weights and biases are initialized from symmetric probability distributions around 0, the dying ReLU NNs occurs in probability as the number of depth goes to infinite. To the best of our knowledge, this is the first theoretical work on the dying ReLU. This result partially explains why training *extremely* deep networks is challenging. Furthermore, it says that the dying ReLU is inevitable as long as the network is deep enough. Also, our result implies that it is the network architecture that decides whether an initialization procedure is good or bad. Our analysis reveals that a specific network architecture can avoid the dying ReLU NNs with high probability. That is, for any $\delta > 0$, when a symmetric initialization is used and $L = \Omega(\log_2(N/\delta))$ is satisfied where L is the number of depth and N is the number of width at each layer, with probability $1 - \delta$, the dying ReLU NNs will not happen. This explains why wide networks are popularly used in practice. Furthermore, it explains why deep and narrow networks have not been successfully used in practice, despite of its representational power [82].

Although there are other approaches to avoid the dying ReLU, we aim to overcome it without changing any network architectures or introducing additional training steps like normalizations. Perhaps, changing the initialization procedure

might be one of the simplest treatments among others. We thus propose a new initialization procedure, namely, a randomized asymmetric initialization (RAI). The new initialization is designed to directly overcome the dying ReLU, while having similar generalization performance to the He initialization [86]. We show that our initialization has a smaller upper bound of the probability of the dying ReLU NNs. All parameters used in our initialization are theoretically chosen to avoid the exploding gradient problem. This is done by the second moment analysis where we derive the expected length map relations between layers [86, 81, 177].

The rest of the chapter is organized as follows. After setting up notation and terminology in Section 2.2, we present the main theoretical results in Section 2.3. In Section 2.4, upon introducing a randomized asymmetric initialization, we discuss its theoretical properties. Numerical examples are provided in Section 2.5, before the conclusion in Section 2.6.

2.2 Mathematical setup

Let $\mathcal{N}^L : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ be a feed-forward neural network with L layers and N_ℓ neurons in the ℓ-th layer ($N_0 = d_{\text{in}}$, $N_L = d_{\text{out}}$). Let us denote the weight matrix and bias vector in the ℓ-th layer by $\boldsymbol{W}^\ell \in \mathbf{R}^{N_\ell \times N_{\ell-1}}$ and $\mathbf{b}^\ell \in \mathbb{R}^{N_\ell}$, respectively. Given an activation function ϕ which is applied element-wise, the feed-forward neural network is recursively defined as follows: $\mathcal{N}^1(\mathbf{x}) = \boldsymbol{W}^1\mathbf{x} + \boldsymbol{b}^1$ and

$$\mathcal{N}^\ell(\mathbf{x}) = \boldsymbol{W}^\ell \phi(\mathcal{N}^{\ell-1}(\mathbf{x})) + \boldsymbol{b}^\ell \in \mathbb{R}^{N_\ell}, \qquad \text{for} \quad 2 \le \ell \le L. \tag{2.1}$$

Here $\mathcal{N}^L$ is called a L-layer neural network or a $(L-1)$-hidden layer neural network. In this chapter, the rectified linear unit (ReLU) activation function is employed, i.e.,

$$\phi(\mathbf{x}) = \mathrm{ReLU}(\mathbf{x}) := (\max\{x_1, 0\}, \cdots, \max\{x_{\text{fan-in}}, 0\}), \text{ where } \mathbf{x} = (x_1, \cdots, x_{\text{fan-in}}).$$

Let $\boldsymbol{\theta}_L = \{\boldsymbol{W}^\ell, \boldsymbol{b}^\ell\}_{1\le\ell\le L}$ be the set of all weight matrices and bias vectors. Let $\mathcal{T}_m = \{\mathbf{x}_i, y_i\}_{1\le i\le m}$ be the set of m training data and let $\mathcal{D} = \{\mathbf{x}_i\}_{i=1}^m$ be the training input data. We assume that $\mathcal{D} \subset B_r(0)$ for some $r > 0$. Given $\mathcal{T}_m$, in order to train $\boldsymbol{\theta}_L$, we consider the standard loss function $\mathcal{L}(\boldsymbol{\theta}_L, \mathcal{T}_m)$:

$$\mathcal{L}(\boldsymbol{\theta}_L, \mathcal{T}_m) = \frac{1}{m} \sum_{(\mathbf{x},y)\in\mathcal{T}_m} \ell(\mathcal{N}^L(\mathbf{x}; \boldsymbol{\theta}_L), y), \tag{2.2}$$

where $\ell : \mathbb{R}^{d_{\text{out}}} \times \mathbb{R}^{d_{\text{out}}} \mapsto \mathbb{R}$ is a loss criterion. In training neural networks, the gradient-based optimization is typically employed to minimize the loss $\mathcal{L}$. The first step for training would be to initialize weight matrices and bias vectors. Typically, they are initialized according to certain probability distributions. For example, uniform distributions around 0 or zero-mean normal distributions are common choices.

In this chapter, we focus on the worst case of dying ReLU, where the entire network dies, i.e., the network becomes a constant function. We refer this as the dying ReLU neural networks. We then define two phases: (1) a network is dead before training, and (2) a network is dead after training. The phase 1 implies the phase 2, but not vice versa. When the phase 1 happens, we say *the network is born dead* (BD).

2.3 Theoretical analysis

In this section, we present a theoretical analysis of the dying ReLU neural networks. We show that a deep ReLU network will eventually be BD in probability as the number of depth L goes to infinity.

Theorem 2.3.1. *Let $\mathcal{N}^L(\boldsymbol{x})$ be a ReLU neural network with L layers, each having N neurons. Suppose that all weights and biases are randomly initialized from probability distributions, which satisfy*

$$P\left(\boldsymbol{W}_j^\ell \in \mathbb{R}_-^N, \boldsymbol{b}_j^\ell < 0\right) \geq p > 0, \qquad \forall 1 \leq j \leq N, \tag{2.3}$$

for some constant $p > 0$, where $\boldsymbol{W}_j^\ell$ is the j-th row of the ℓ-th layer weight matrix and $\boldsymbol{b}_j^\ell$ is the j-th component of the ℓ-th layer bias vector. Then

$$\lim_{L\to\infty} P\left(\mathcal{N}^L(\boldsymbol{x};\boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) = 1,$$

where $\mathcal{D}$ is the training input data.

Proof. The proof can be found in Appendix A.1. □

We remark that Equation 2.3 is a very mild condition and it can be satisfied in many cases. For example, when symmetric probability distributions around 0 are employed, the condition is met with $p = 2^{-N-1}$. Theorem 2.3.1 implies that the fully connected ReLU network will be dead at the initialization as long as the network is deep enough. This explains theoretically why training a very deep network is hard.

Theorem 2.3.1 shows that the ReLU network asymptotically will be dead. Thus, we are now concerned with the convergence behavior of the probability of NNs being BD. Since almost all common initialization procedures use symmetric probability distributions around 0, we derive an upper bound of the born dead probability (BDP) for symmetric initialization.

Theorem 2.3.2. *Let $\mathcal{N}^L(\boldsymbol{x})$ be a ReLU neural network with L layers, each having $N_1, \cdots, N_L$ neurons. Suppose that all weights are independently initialized from symmetric probability distributions around 0 and all biases are either drawn from a symmetric distribution or set to zero. Then*

$$P\left(\mathcal{N}^L(\boldsymbol{x};\boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) \leq 1 - \prod_{\ell=1}^{L-1}(1-(1/2)^{N_\ell}), \tag{2.4}$$

where $\mathcal{D}$ is the training input data. Furthermore, assuming $N_\ell = N$ for all ℓ,

$$\lim_{L\to\infty} P\left(\mathcal{N}^L(\boldsymbol{x};\boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) = 1,$$
$$\lim_{N\to\infty} P\left(\mathcal{N}^L(\boldsymbol{x};\boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) = 0.$$

Proof. The proof can be found in Appendix A.2. □

Theorem 2.3.2 provides an upper bound of the BDP. It shows that at a fixed depth L, the network will not be BD in probability as the number of width N goes to infinite. In order to understand how this probability behaves with respect to the number of width and depth, a lower bound is needed. We thus provide a lower bound of the BDP of ReLU NNs at $d_{\text{in}} = 1$.

Theorem 2.3.3. *Let $\mathcal{N}^L(\boldsymbol{x})$ be a bias-free ReLU neural network with $L \geq 2$ layers, each having N neurons at $d_{in} = 1$. Suppose that all weights are independently initialized from*

continuous symmetric probability distributions around 0, which satisfies

$$P(\langle \boldsymbol{W}_j^\ell, \boldsymbol{v}_1\rangle > 0, \langle \boldsymbol{W}_j^\ell, \boldsymbol{v}_2\rangle < 0 | \boldsymbol{v}_1, \boldsymbol{v}_2) \leq \frac{1}{4}, \quad \mathbf{0} \neq \boldsymbol{v}_1, \boldsymbol{v}_2 \in \mathbb{R}_+^N, \quad \forall 1 \leq j \leq N,$$

where $\boldsymbol{W}_j^\ell$ *is the* j*-th row of the* ℓ*-th layer weight matrix. Then*

$$p_{low}(L, N) \leq P\left(\boldsymbol{\mathcal{N}}^L(\boldsymbol{x}; \boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) \leq 1 - \prod_{\ell=1}^{L-1}(1 - (1/2)^N),$$

where $a_1 = 1 - (1/2)^N$, $a_2 = 1 - (1/2)^{N-1} - (N-1)(1/4)^N$, *and*

$$p_{low}(L, N) = 1 - a_1^{L-2} + \frac{(1 - 2^{-N+1})(1 - 2^{-N})}{1 + (N-1)2^{-N}}(-a_1^{L-2} + a_2^{L-2}).$$

Proof. The proof can be found in Appendix A.3. □

Theorem 2.3.3 reveals that the BDP behavior depends on the network architecture. In Fig. 2.2, we plot the BDP with respect to increasing the number of layers at varying width from $N = 2$ to $N = 5$. A bias-free ReLU feed-forward NN with $d_{\text{in}} = 1$ is employed with weights randomly initialized from symmetric distributions. The results of one million independent simulations are used to calculate each probability estimation. Numerical estimations are shown as symbols. The upper and lower bounds from Theorem 2.3.3 are also plotted with dash and dash-dot lines, respectively. We see that when the NN gets narrower, the probability of NN being BD grows faster as the depth increases. Also, at a fixed width N, the BDP grows as the number of layer increases. This is expected by Theorems 2.3.1 and 2.3.3.

Once the network is BD, we have no hope to train the network successfully. Here we provide a formal statement of the consequence of the network being BD.

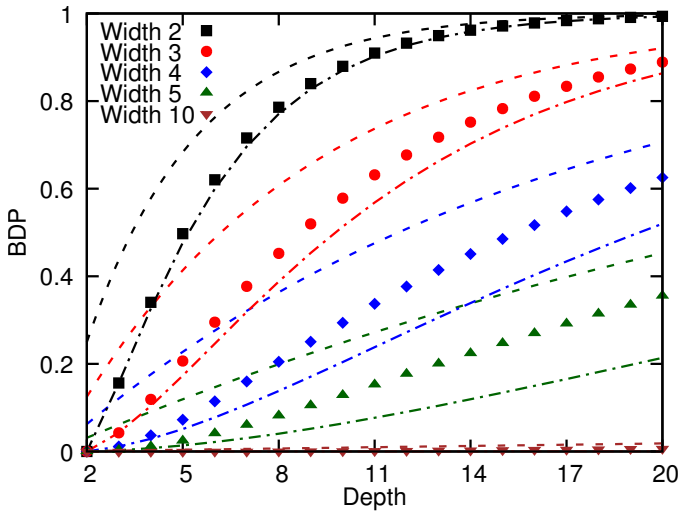


Figure 2.2: Probability of a ReLU NN to be born dead as a function of the number of layers for different widths. The dash lines represent the upper and lower bounds from Theorem 2.3.3. The symbols represent our numerical estimations. Similar colors correspond to the same width.

Theorem 2.3.4. *Suppose that the feed-forward ReLU neural network is BD. Then, for any loss function $\mathcal{L}$, and for any gradient based method, the ReLU network is optimized to be a constant function, which minimizes the loss.*

Proof. The proof can be found in Appendix A.4. □

Theorem 2.3.4 implies that no matter what gradient-based optimizers are employed including stochastic gradient desecent (SGD), SGD-Nesterov [208], AdaGrad [57], AdaDelta [231], RMSProp [89], Adam [112], BFGS [167], L-BFGS [29], the network is trained to be a constant function which minimizes the loss.

If the online-learning or the stochastic gradient method is employed, where the training data are independently drawn from a probability distribution $P_{\mathcal{D}}$, the optimized network is

$$\mathcal{N}^L(\mathbf{x};\boldsymbol{\theta}^*) = \boldsymbol{c}^* = \arg\min_{\boldsymbol{c}\in\mathbb{R}^{N_L}} \mathbb{E}\left[\ell(\boldsymbol{c}, f(\mathbf{x}))\right],$$

where the expectation $\mathbb{E}$ is taken with respect to $\mathbf{x} \sim P_{\mathcal{D}}$. For example, if L^2-loss is employed, i.e., $\ell(\mathcal{N}^L(\mathbf{x}), f(\mathbf{x})) = (\mathcal{N}^L(\mathbf{x}) - f(\mathbf{x}))^2$, the resulting network is $\mathbb{E}[f(\mathbf{x})]$. If L^1 loss is employed, i.e., $\ell(\mathcal{N}^L(\mathbf{x}), f(\mathbf{x})) = |\mathcal{N}^L(\mathbf{x}) - f(\mathbf{x}))|$, the resulting network is the median of $f(\mathbf{x})$ with respect to $\mathbf{x} \sim P_{\mathcal{D}}$. Note that the mean absolute error (MAE) and the mean squared error (MSE) used in practice are discrete versions of L^1 and L^2 loss, respectively, if the size of minibatch is large.

When we design a neural network, we want the BDP to be small, say, less than 1% or 10%. Then, the upper bound (Equation 2.4) of Theorem 2.3.2 can be used for designing a specific network architecture, which has a small probability of NNs being born dead.

Corollary 2.3.5. *Suppose $N_\ell = N$ for all ℓ. For fixed depth L and $\delta > 0$, if the width N is $N = \log_2 L/\delta$, with probability exceeding $1 - \delta$, the ReLU neural network will not be initialized to be dead in $\mathcal{D}$.*

Proof. This readily follows from

$$P\left(\mathcal{N}^L(\mathbf{x}; \boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) \leq 1 - (1 - 2^{-N})^{L-1}$$
$$\leq 1 - (1 - (L-1)2^{-N}) \leq L2^{-N} = \delta.$$

□

As a practical guide, we constructed a diagram shown in Fig. 2.3 that includes both theoretical predictions and our numerical tests. We see that as the number of layers increases, the numerical tests match closer the theoretical results. It is clear from the diagram that a 10-layer NN of width 10 has a probability of dying less than 1% whereas a 10-layer NN of width 5 has a probability of dying greater than

10%; for width of three the probability is about 60%. Note that the growth rate of the maximum number of layers is exponential which is expected by Corollary 2.3.5.

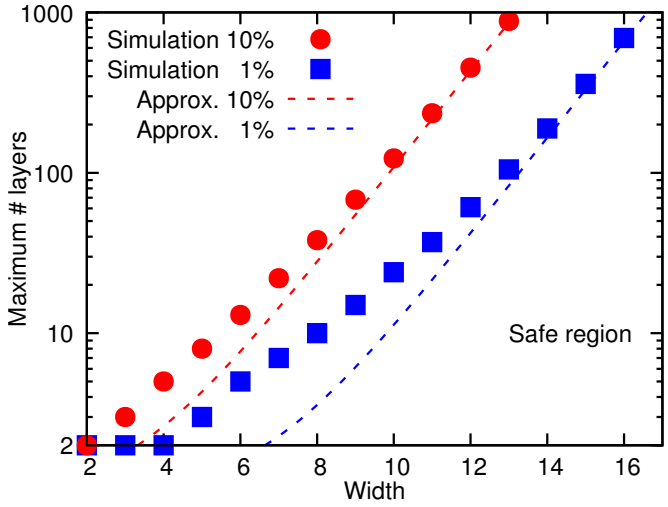


Figure 2.3: Diagram indicating safe operating regions for a ReLU NN. The dash lines represent Corollary 2.3.5 while the symbols represent our numerical tests. The maximum number of layers of a neural network can be used at different width to keep the probability of collapse less than 1% or 10%. The region below the blue line is the safe region when we design a neural network. As the width increases the theoretical predictions match closer with our numerical simulations.

Our results explain why deep and narrow networks have not been popularly employed in practice and reveal a limitation of training extremely deep networks. We remark that modern practical networks happen to be sufficiently wide enough to have small BDPs. We believe that the born dead network problem is a major impediment for one to explore a potential use of deep and narrow networks.

2.4 Randomized asymmetric initialization

The so-called 'He initialization' [86] is perhaps one of the most popular initialization schemes in deep learning community, especially when the ReLU activation function is concerned. The effectiveness of the He initialization has been shown in many machine learning applications. The He initialization uses mean zero normal

distributions. Thus, as we discussed earlier, it suffers from the dying ReLU. We thus propose a new initialization procedure, namely, a randomized asymmetric initialization. The motivation is in twofolds. One is to mimics the He initialization so that the new scheme can produce similar generalization performance. The other is to alleviate the problem of dying ReLU neural networks. We hope that the proposed initialization opens a new way to explore the potential use of deep and narrow networks in practice.

For ease of discussion, we introduce some notation. For any vector $\boldsymbol{v} \in \mathbb{R}^{n+1}$ and $k \in \{1, \cdots, n+1\}$, we define

$$\boldsymbol{v}_{-k} = (v_1, \cdots, v_{k-1}, v_{k+1}, \cdots, v_{n+1})^T \in \mathbb{R}^n. \tag{2.5}$$

In order to train a L-layer neural network, we need to initialize $\boldsymbol{\theta}_L = \{\boldsymbol{W}^\ell, \boldsymbol{b}^\ell\}_{1 \le \ell \le L}$. At each layer, let $\mathbf{V}^\ell = [\boldsymbol{W}^\ell, \boldsymbol{b}^\ell] \in \mathbb{R}^{N_\ell \times (N_{\ell-1}+1)}$. We denote the j-th row of $\mathbf{V}^\ell$ by $\mathbf{V}_j^\ell = [\boldsymbol{W}_j^\ell, \boldsymbol{b}_j^\ell] \in \mathbb{R}^{N_{\ell-1}+1}$, $j = 1, \cdots, N_\ell$ where $N_0 = d_{\text{in}}$ and $N_L = d_{\text{out}}$.

2.4.1 Proposed initialization

We propose to initialize $\mathbf{V}^\ell$ as follows. Let P_ℓ be a probability distribution defined on $[0, M_\ell]$ for some $M_\ell > 0$ or $[0, \infty)$. Note that P_ℓ is asymmetric around 0. At the first layer of $\ell = 1$, we employ the 'He initialization' [86], i.e., $\mathbf{W}_{ij}^1 \sim N(0, 2/d_{\text{in}})$ and $\mathbf{b}^1 = \mathbf{0}$. For $\ell \ge 2$, and each $1 \le j \le N_\ell$, we initialize $\boldsymbol{V}_j^\ell$ as follows:

1. Randomly choose k_j^ℓ in $\{1, 2, \cdots, N_{\ell-1}, N_{\ell-1}+1\}$.
2. Initialize $(\mathbf{V}_j^\ell)_{-k_j^\ell} \sim \boldsymbol{\mathcal{N}}(0, \sigma_\ell^2 \boldsymbol{I})$ and $(\mathbf{V}_j^\ell)_{k_j^\ell} \sim \mathrm{P}_\ell$.

Since an index is randomly chosen at each ℓ and j and a positive number is randomly drawn from an asymmetric probability distribution around 0, we name this new initialization a randomized asymmetric initialization. Only for the first layer, the He initialization is employed. This is because since an input could have a negative value, if the weight which corresponds to the negative input were to be initialized from P_ℓ, this could cause the dying ReLU. We note that the new initialization requires us to choose σ_ℓ^2 and P_ℓ. In Subsection 2.4.2, these will be theoretically determined. One could choose multiple indices in the step 1 of the new initialization. However, for simplicity, we constraint ourselves to a single index case.

We first show that this new initialization procedure results in a smaller upper bound of the BDP.

Theorem 2.4.1. *If a ReLU feed-forward neural network $\mathcal{N}^L$ with L layers, each having width $N_1, \cdots, N_L$, is initialized by the randomized asymmetric initialization, then*

$$P\left(\mathcal{N}^L(\boldsymbol{x};\boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) \leq 1 - \prod_{\ell=1}^{L-1}\left(1 - (1/2 - \gamma_\ell)^{N_\ell}\right),$$

where $\gamma_1 = 0$ and γ_j's are some constants in $(0, 0.5]$, which depend on $\{N_\ell\}_{\ell=1}^{L-1}$ and the training input data $\mathcal{D}$.

Proof. The proof can be found in Appendix A.5. □

When a symmetric initialization is employed, $\gamma_j = 0$ for all $1 \leq j < N_L$, which results in Equation 2.4 of Theorem 2.3.2. Although the new initialization has a smaller upper bound compared to those by symmetric initialization, as Theorem 2.3.1 suggests, it also asymptotically suffers from the dying ReLU.

Corollary 2.4.2. *Assuming the same conditions in Theorem 2.4.1, and $N_\ell = N$ for all ℓ. Then, there exists $2 < \gamma$, which depends on N, L and the training input data $\mathcal{D}$, such that*

$$P\left(\mathcal{N}^L(\boldsymbol{x};\boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) \leq 1 - \prod_{\ell=1}^{L-1}\left(1-(1/\gamma)^N\right).$$

For fixed depth L and $\delta > 0$, if the width N is $N = \log_\gamma L/\delta$, with probability exceeding $1-\delta$, the ReLU neural network will not be initialized to be dead. Furthermore,

$$\lim_{L\to\infty} P\left(\mathcal{N}^L(\boldsymbol{x};\boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) = 1,$$
$$\lim_{N\to\infty} P\left(\mathcal{N}^L(\boldsymbol{x};\boldsymbol{\theta}_L) \text{ is born dead in } \mathcal{D}\right) = 0.$$

Proof. The proof is readily followed from Theorem 2.3.1, 2.4.1 and Corollary 2.3.5. □

2.4.2 Second moment analysis

The proposed randomized asymmetric initialization described in Subsection 2.4.1 requires us to determine σ_ℓ^2 and P_ℓ. Similar to the He initialization [87], we aim to properly choose initialization parameters from the length map analysis. Following the work of [177], we present the analysis of a single input propagation through the deep ReLU network. To be more precise, we track the expectation of the normalized squared length of the input vector at each layer, $\mathbb{E}[q^\ell(\mathbf{x})]$, where $q^\ell(\mathbf{x}) = \frac{\|\mathcal{N}^\ell(\mathbf{x})\|^2}{N_\ell}$. The expectation $\mathbb{E}$ is taken with respect to all weights and biases.

Theorem 2.4.3. *Let P_ℓ be a probability distribution whose support is $[0, M_\ell] \subset \mathbb{R}^+$. Let $X_\ell \sim P_\ell$ have finite first and second moments, i.e., $\mu'_{\ell,i} = E[X_\ell^i] < \infty$, for $i = 1, 2$, and $\mu'_{\ell,1} \geq M_\ell/2$. Suppose the ℓ-th layer weights and biases are initialized by the randomized*

asymmetric initialization described in Subsection 2.4.1. Then for any input $\boldsymbol{x} \in \mathbb{R}^{d_{in}}$*, we have*

$$\frac{\mathcal{A}_{low,\ell}}{2}\mathbb{E}[q^{\ell}(\boldsymbol{x})] + \sigma_{b,\ell}^2 \leq \mathbb{E}[q^{\ell+1}(\boldsymbol{x})] \leq \frac{\mathcal{A}_{upp,\ell}}{2}\mathbb{E}[q^{\ell}(\boldsymbol{x})] + \sigma_{b,\ell}^2,$$

where $\sigma_{b,\ell}^2 = \frac{\mu'_{\ell+1,2}+\sigma_{\ell+1}^2 N_{\ell+1}}{N_{\ell+1}+1}$, $\mathcal{A}_{low,\ell} = \frac{\sigma_{b,\ell+1}^2}{\sigma_{b,\ell}^2}\left(\frac{N_\ell \mu'_{\ell,2}+N_{\ell-1}\sigma_w^2}{N_{\ell-1}+1}\right)$, *and*

$$\mathcal{A}_{upp,\ell} = \frac{\sigma_{b,\ell+1}^2}{\sigma_{b,\ell}^2}\left(\frac{N_{\ell-1}\sigma_w^2 + 2\sqrt{2/\pi}N_\ell \mu'_{\ell,1}\sigma_w + 2N_\ell \mu'_{\ell,2}}{N_{\ell-1}+1}\right).$$

Proof. The proof can be found in Appendix A.6. □

Corollary 2.4.4. *Under the same conditions of Theroem 2.4.3, if* $N_\ell = N$, $M_\ell = M$, $E[X_\ell^i] = \mu'_i$, $i = 1, 2$ *for all* ℓ*, and* $\mu'_1 \geq M/2$*, we have*

$$\frac{\mathcal{A}_{low,\ell}}{2}E[q^{\ell}(\boldsymbol{x})] + \sigma_{b,\ell}^2 \leq E[q^{\ell+1}(\boldsymbol{x})] \leq \frac{\mathcal{A}_{upp,\ell}}{2}E[q^{\ell}(\boldsymbol{x})] + \sigma_{b,\ell}^2,$$

where $\sigma_{b,\ell}^2 = \frac{\mu'_2+\sigma_w^2}{N+1}$, $\mathcal{A}_{low,\ell} = \frac{N(\mu'_2+\sigma_w^2)}{N+1}$, *and* $\mathcal{A}_{upp,\ell} = \frac{N(\sigma_w^2+2\sqrt{2/\pi}\mu'_1\sigma_w+2\mu'_2)}{N+1}$.

Since $\sigma_{b,\ell}^2 > 0$, $\lim_{\ell\to\infty} E[q^{\ell}(\mathbf{x})]$ cannot be zero. In order for $\lim_{\ell\to\infty} E[q^{\ell}(\mathbf{x})] < \infty$, the initialization parameters $(\sigma_w^2, \mu'_{\ell,1}, \mu'_{\ell,2})$ have to be chosen to satisfy $\mathcal{A}_{upp,\ell} < 2$. Assuming $\mu'_2 < 1$, if σ_w is chosen to be

$$\sigma_w = \sqrt{2}\left(-\frac{\mu'_1}{\sqrt{\pi}} + \sqrt{\frac{\mu_1'^2}{\pi} + 1 - \mu'_2}\right), \tag{2.6}$$

we have $\frac{\mathcal{A}_{upp,\ell}}{2} = \frac{N}{N+1}$ which satisfies the condition.

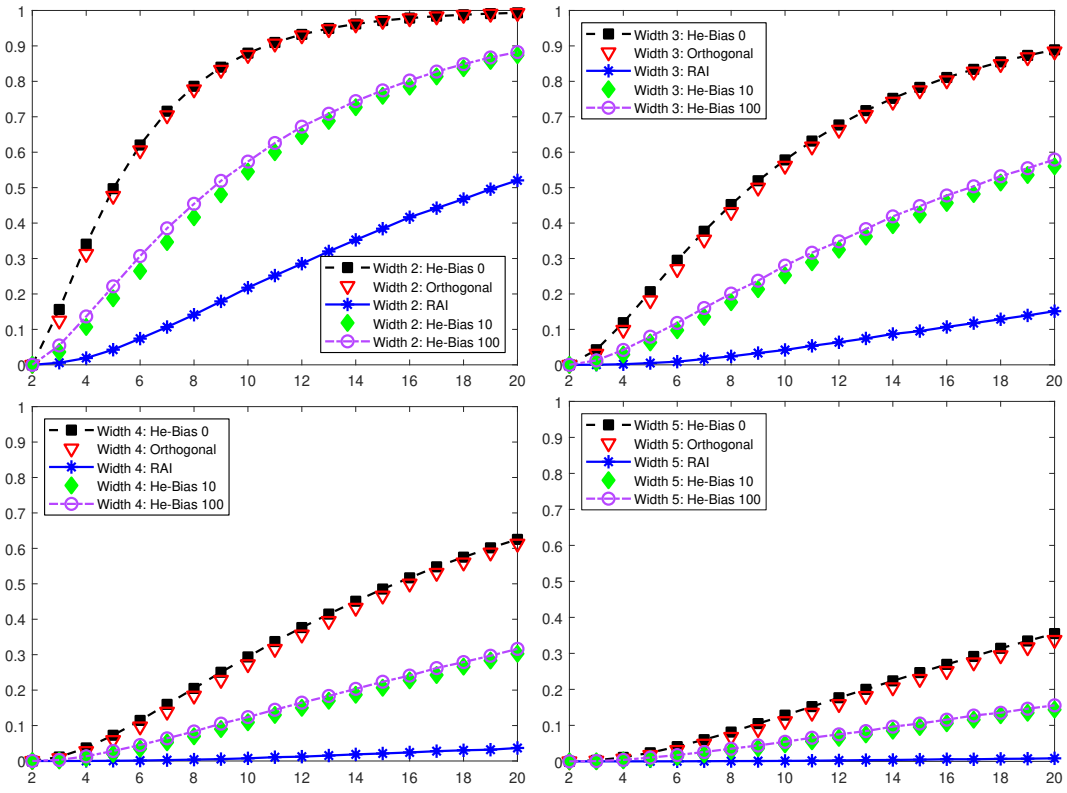


Figure 2.4: The BDPs are plotted with respect to increasing the number of depth L at varying width $N = 2$ (top left), $N = 3$ (top right), $N = 4$ (bottom left), and $N = 5$ (bottom right). The ReLU neural networks in $d_{\text{in}} = 1$ are employed. The square, diamond and circle symbols correspond to the He initialization [86] with constant bias 0, 10 and 100, respectively. The inverted triangle symbols correspond to the orthogonal initialization [192]. The asterisk symbols correspond to the proposed randomized asymmetric initialization (RAI).

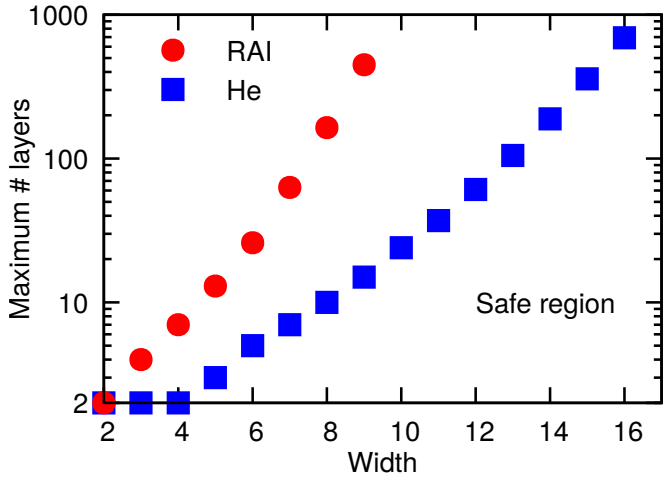


Figure 2.5: Comparison of the safe operating regions for a ReLU NN between RAI and He initializations. The maximum number of layers of a neural network can be used at different width to keep the probability of collapse less than 1%.

2.4.3 Comparison against other initialization procedures

In Fig. 2.4, we demonstrate the probability that the network is BD by the proposed randomized asymmetric initialization (RAI) method. Here we employ $\mathrm{P} = \mathrm{Beta}(2, 1)$ and $\sigma_w = -\frac{2\sqrt{2}}{3\sqrt{\pi}} + \sqrt{1 + \frac{8}{9\pi}} \approx 0.6007$ from Equation 2.6 (see Appendix A.7 for the Python code). To compare against other procedures, we present the results by the He initialization [86]. We also present the results of existing asymmetric initialization procedures; the orthogonal [192] and the layer-sequential unit-variance (LSUV) [149] initializations. The LSUV is the orthogonal initialization combined with rescaling of weights such that the output of each layer has unit variance. Because weight rescaling cannot make the output escape from the negative part of ReLU, it is sufficient to consider the orthogonal initialization. We see that the BDPs by the orthogonal initialization are very close to and a little lower than those by the He initialization. This implies that the orthogonal initialization cannot prevent the dying ReLU network. Furthermore, we show the results by the He initialization with positive constant bias of 10 and 100. Naively speaking, having a big positive bias will help in preventing dying ReLU neurons, as the input of each layer is pushed to be positive, although this might cause the exploding gradient problem in training. We see that the BDPs by the He with bias 10 and 100 are lower than those by the He with bias 0 and the orthogonal initialization. However, it is clearly observed that our proposed initialization (RAI) drastically drops the BDPs compared to all others. This is implied by Theorem 2.4.1.

As a practical guide, we constructed a diagram shown in Fig. 2.5 to demonstrate the 1% safe operation region by the RAI. It is clear from the diagram that the RAI drastically expands its 1% safe region compared to those by the He initialization. Note that the growth rate of the maximum number of layers is exponential which

is expected by Corollary 2.4.2.

2.5 Numerical examples

We demonstrate the effectiveness of the proposed randomized asymmetric initialization (RAI) in training deep ReLU networks.

Test functions include one- and two-dimensional functions of different regularities. The following test functions are employed as unknown target functions. For one dimensional cases,

$$f_1(x) = |x|, \qquad f_2(x) = x\sin(5x), \qquad f_3(x) = 1_{\{x>0\}}(x) + 0.2\sin(5x). \tag{2.7}$$

For two dimensional case,

$$f_4(x_1, x_2) = \begin{bmatrix} |x_1 + x_2| \\ |x_1 - x_2| \end{bmatrix}. \tag{2.8}$$

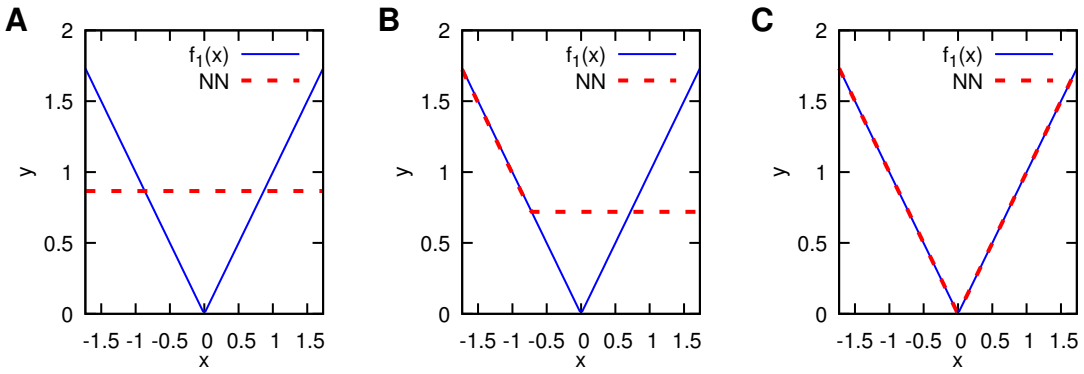
We employ the network architecture having the width of $d_{\text{in}} + d_{\text{out}}$ at all layers. Here d_{in} and d_{out} are the dimensions of the input and output, respectively. It was shown in [82] that the minimum number of width required for the universal approximation is less than or equal to $d_{\text{in}} + d_{\text{out}}$. We thus choose this specific network architecture. as it theoretically guarantees to approximate any continuous function. In all numerical examples, we employ one of the most popular first-order gradient-based optimization, `Adam` [112] with the default parameters. The minibatch size is chosen to be either 64 or 128. The standard L_2-loss function $\ell(\mathcal{N}^L(\mathbf{x}, \boldsymbol{\theta}), (\mathbf{x}, y)) = (\mathcal{N}^L(\mathbf{x}; \boldsymbol{\theta}) - y)^2$ is used on 3,000 training data. The training

data are randomly uniformly drawn from $[-\sqrt{3}, \sqrt{3}]^{d_{\text{in}}}$. Without changing any setups described above, we present the approximation results based on different initialization procedures. The results by our proposed randomized asymmetric initialization are referred to 'Rand. Asymmetric' or 'RAI'. Specifically, we use $\mathrm{P} = \mathrm{Beta}(2, 1)$ with σ_w defined in Equation 2.6. To compare against other methods, we also show the results by the He initialization [86]. We present the ensemble of 1,000 independent training simulations.

In one dimensional examples, we employ a 10-layer ReLU network of width 2. It follows from Fig. 2.4 that we expect to observe at least 88% training results by the symmetric initialization and 22% training results by the RAI are collapsed. In the two dimensional example, we employ a 20-layer ReLU network of width 4. According to Fig. 2.4, we expect to see at least 63% training results by the symmetric initialization and 3.7% training results by the RAI are collapsed.

Fig. 2.6 shows all training outcomes of our 1,000 simulations for approximating $f_1(x)$ and its corresponding empirical probabilities by different initialization schemes. For this specific test function, we observe only 3 trained results shown in A, B, C. In fact, $f_1(x)$ can be represented exactly by a 2-layer ReLU network with width 2, $f_1(x) = |x| = \mathrm{ReLU}(x) + \mathrm{ReLU}(-x)$. It can clearly be seen that the He initialization results in the collapse with probability more than 90%. However, this probability is drastically reduced to 40% by the RAI. These probabilities are different from the probability that the network is BD. This implies that even though the network wasn't BD, there are cases that after training, the network dies. In this example, 5.6% and 18.3% of results by the symmetric and our method, respectively, are not dead at the initialization, however, they are ended up with collapsing after training. The 37.3% of training results by the RAI perfectly recover the target function $f_1(x)$, however, only 2.2% of results by the He initialization achieve this

success. Also, 22.4% of the RAI and 4.2% of the He initialization produce the half-trained results which correspond to Fig. 2.6 (B). We remark that the only difference in training is the initialization. This implies that our new initialization does not only prevent the dying ReLU network but also improves the quality of the training in this case.

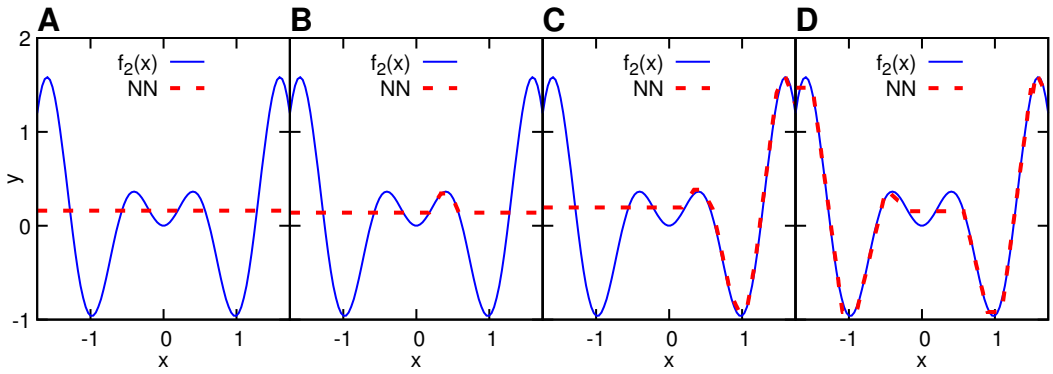


$f_1 x$	Collapse (A)	Half-Trained (B)	Success (C)
Symmetric (He init.)	93.6%	4.2%	2.2%
Rand. Asymmetric	**40.3%**	**22.4%**	**37.3%**

Figure 2.6: The approximation results for $f_1(x)$ using a 10-layer ReLU network of width 2. For this specific test function, we observe only 3 trained results shown in A, B, C. The table shows the corresponding empirical probabilities from 1,000 independent simulations. The only difference is the initialization.

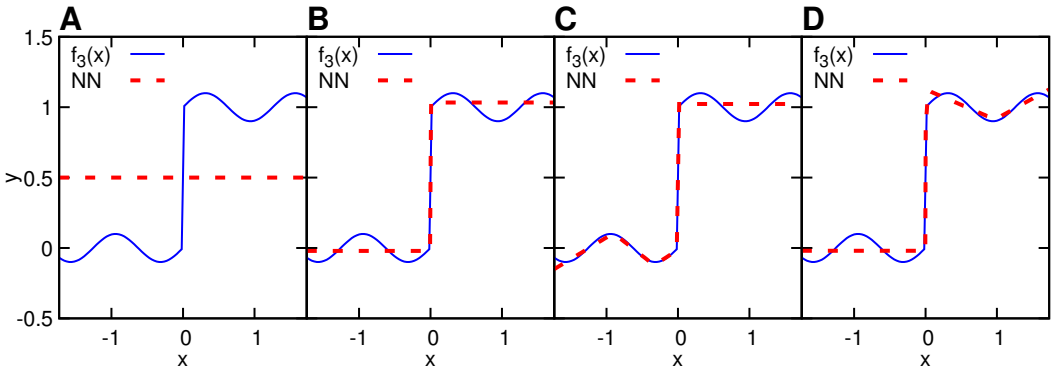
The approximation results for $f_2(x)$ are shown in Fig. 2.7. Note that f_2 is a C^∞ function. It can be seen that 91.9% of training results by the symmetric initialization and 29.2% of training results by the RAI are collapsed which correspond to Fig. 2.7 (A). This indicates that the RAI can effectively alleviate the dying ReLU. In this example, 3.9% and 7.2% of results by the symmetric and our method, respectively, are not dead at the initialization, however, they are ended up with collapsing after training. Except for the collapse, other training outcomes are not easy to be classified. Fig. 2.7 (B,C,D) show three training results among many others. We observe that the behavior and result of training are not easily predictable in general. However, we consistently observe partially collapsed results after training. Such

partial collapses are also observed in Fig. 2.7 (B,C,D). We believe that this requires more attention and postpone the study of this partial collapse to future work.



$f_2 x$	Collapsed (A)	Not collapsed (B,C,D)
Symmetric (He init.)	91.9%	8.1%
Rand. Asymmetric	**29.2%**	**70.8%**

Figure 2.7: The approximation results for $f_2(x)$ using a 10-layer ReLU network of width 2. Among many trained results, four are shown. The table shows the corresponding empirical probabilities from 1,000 independent simulations. The only difference is the initialization.



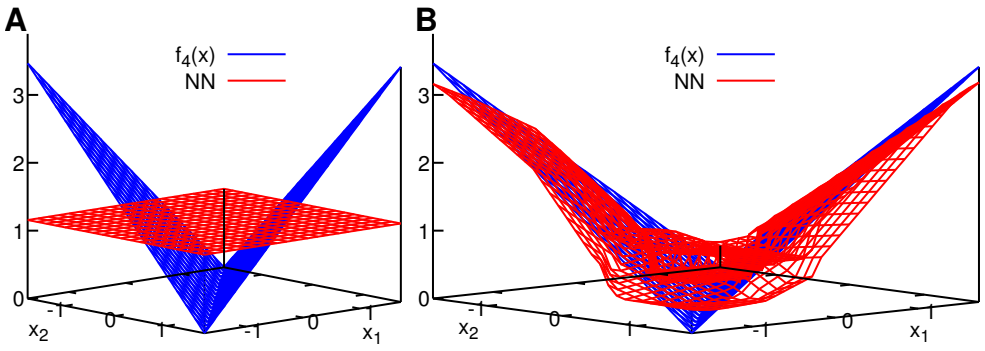
$f_3 x$	Collapsed (A)	Not collapsed (B,C,D)
Symmetric (He init.)	93.8%	6.2%
Rand. Asymmetric	**32.6%**	**67.4%**

Figure 2.8: The approximation results for $f_3(x)$ using a 10-layer ReLU network of width 2. Among many trained results, four are shown. The table shows the corresponding empirical probabilities from 1,000 independent simulations. The only difference is the initialization.

Similar behavior is observed for approximating a discontinuous function $f_3(x)$. The approximation results for $f_3(x)$ and its corresponding empirical probabilities are shown in Fig. 2.8. We see that 93.8% of training results by the He initialization and 32.6% of training results by the RAI are collapsed which correspond to Fig. 2.8 (A). In this example, the RAI drops the probability of collapsing by 60.3 percentage

point. Again, this implies that the RAI can effectively avoid the dying ReLU, especially when deep and narrow ReLU networks are employed. Fig. 2.8 (B,C,D) show three trained results among many others. Again, we observe partially collapsed results.

Next we show the approximation result for a multi-dimensional inputs and outputs function $f_4(\mathbf{x})$ defined in Equation 2.8. We observe similar behavior. Fig. 2.9 shows some of approximation results for f_4 and its corresponding probabilities. Among 1,000 independent simulations, the collapsed results are obtained by the He initialization with 76.8% probability and by the RAI with 9.6% probability. From Fig. 2.4, we expect to observe at least 63% and 3.7% of results by the symmetric and the RAI to be collapsed. Thus, in this example, 13.8% and 5.9% of results by the symmetric and our method, respectively, are not dead at the initialization, however, they are ended up with Fig. 2.9 (A) after training. This indicates that the RAI can also effectively overcome the dying ReLU in multi-dimensional inputs and outputs tasks.



$f_4 x$	Collapsed (A)	Not collapsed (B)
Symmetric (He init.)	76.8%	23.2%
Rand. Asymmetric	**9.6%**	**90.4%**

Figure 2.9: The approximation results for $f_4(\mathbf{x})$ using a 20-layer ReLU network of width 4. Among many trained results, two are shown. The table shows the corresponding empirical probabilities from 1,000 independent simulations. The only difference is the initialization.

As a last example, we demonstrate the performance of the RAI on the MNIST

dataset. For the training, we employ the cross-entropy loss and the mini-batch size of 100. The networks are trained using `Adam` [112] with its default values. In Fig. 2.10, the convergence of the test accuracy is shown with respect to the number of epochs. On the left and right, we employ the ReLU network of depth 2 and width 1024 and of depth 50 and with 10, respectively. We see that when the shallow and wide network is employed, the RAI and the He initialization show similar generalization performance (test accuracy). However, when the deep and narrow network is employed, the RAI performs better than the He initialization. This indicates that the proposed RAI not only reduces the BDP of deep networks, but also has good generalization performance.

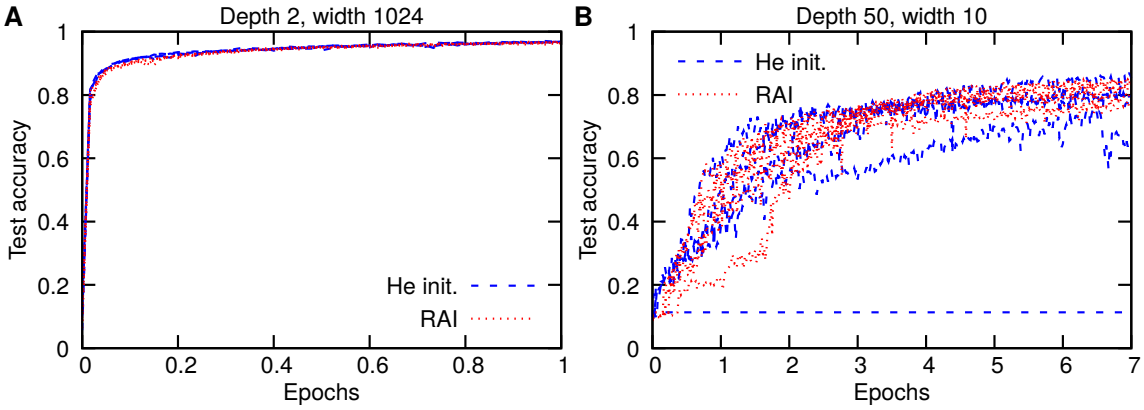


Figure 2.10: The test accuracy on the MNIST of five independent simulations are shown with respect to the number of epochs by the He initialization and the RAI. (Left) A shallow (depth 2, width 1024) ReLU network is employed. The He initialization and the RAI have similar performance. (Right) A deep (depth 50, width 10) ReLU network is employed. The RAI results in higher test accuracy than the He initialization.

2.6 Conclusion

In this chapter, we establish, to the best of our knowledge, the first theoretical analysis on the dying ReLU. By focusing on the worst case of dying ReLU, we define 'the dying ReLU network' which refers to the problem when the ReLU

network is dead. We categorize the dying process into two phases. One phase is the event where the ReLU network is initialized to be a constant function. We refer to this event as 'the network is born dead'. The other phase is the event where the ReLU network is collapsed after training. Certainly, the first phase implies the second, but not vice versa. We show that the probability that the network is born dead goes to 1 as the depth goes infinite. Also, we provide an upper and a lower bound of the dying probability in $d_{\text{in}} = 1$ when the standard symmetric initialization is used.

Furthermore, in order to overcome the dying ReLU networks, we propose a new initialization procedure, namely, a randomized asymmetric initialization (RAI). We show that the RAI has a smaller upper bound of the probability of NNs being born dead. By establishing the expected length map relation (second moment analysis), all parameters needed for the new method are theoretically designed. Numerical examples are provided to demonstrate the performance of our method. We observe that the RAI does not only overcome the dying ReLU but also improves the training and generalization performance.

Chapter

THREE

Quantifying the Generalization Error in Deep Learning in terms of Data Distribution and Neural Network Smoothness

Manuscript information

A version of this chapter appeared in [101]: P. Jin*, L. Lu*, Y. Tang, & G. E. Karniadakis. Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness. *arXiv preprint arXiv:1905.11427*, 2019.

Author contributions: P.J. and L.L. developed the theory and performed the experiments. P.J., L.L., and G.E.K. wrote the manuscript. Y.T. and G.E.K. supervised the project.

3.1 Introduction

In the last 15 years, deep learning, i.e., deep neural networks (NNs), has been used very effectively in diverse applications, such as image classification [115], natural language processing [140], and game playing [195]. Despite this remarkable success, our theoretical understanding of deep learning is lagging behind. The accuracy of NNs can be characterized by dividing the expected error into three main types: approximation (also called expressivity), optimization, and generalization [26, 25], see Fig. 3.1. The well-known universal approximation theorem was obtained by [49] and [91] almost three decades ago stating that feed-forward neural nets can approximate essentially any function if their size is sufficiently large. In the past several years, there have been numerous studies that analyze the landscape of the non-convex objective functions, and the optimization process by stochastic gradient descent (SGD) [126, 131, 5, 55, 137]. Whereas there are some satisfactory answers to the problems of approximation and optimization, much

less is known about the theory of generalization, which is the focus of this study.

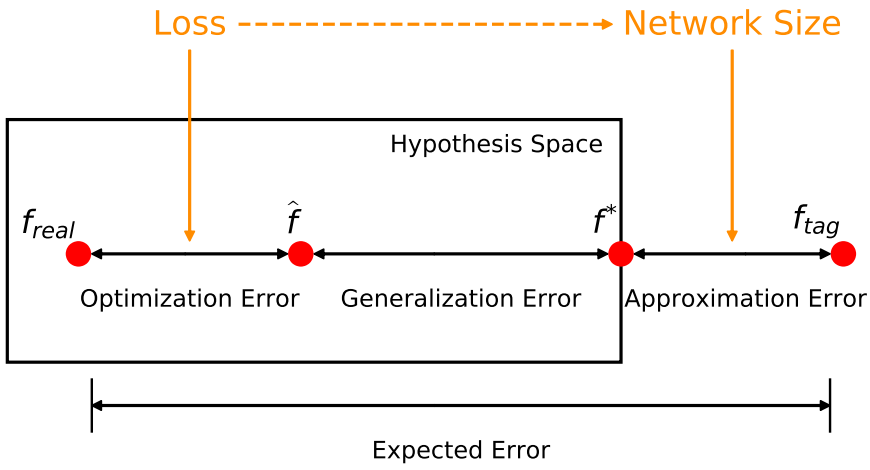


Figure 3.1: Illustration of approximation error, optimization error, and generalization error. The total error consists of these three errors. f_{tag} is the target true function, f^* is the function closest to f_{tag} in the hypothesis space, $\hat{f}$ is a neural network whose loss is at a global minimum of the empirical loss, and f_{real} is the function returned by the training algorithm. Thus, the optimization error is correlated with the value of the empirical loss, while the approximation error depends on the network size. In addition, a small loss requires a large network size, which in turn leads to a small approximation error. Assuming a sufficiently small empirical loss, the expected error mainly depends on the generalization error.

The classical analysis of generalization is based on controlling the complexity of the function class, i.e., model complexity, by managing the bias-variance trade-off [66]. However, this type of analysis is not able to explain the small generalization gap between training and test performance of neural networks learned by SGD in practice, considering the fact that deep neural networks often have far more model parameters than the number of samples they are trained on, and have sufficient capacity to memorize random labels [164, 232]. To explain this phenomenon, several approaches have been recently developed by many researchers. The first approach is characterizing neural networks with some other low "complexity" instead of the traditional Vapnik-Chervonenkis (VC) dimension [14] or Rademacher complexity [15], such as path-norm [163], margin-based bounds [198, 13, 161], Fisher-Rao norm [130], and more [162, 221]. The second approach is to analyze

some good properties of SGD or its variants, including its stability [84, 117, 73, 39], robustness [197, 198], implicit biases/regularization [175, 199, 77, 153], and the structural properties (e.g., sharpness) of the obtained minimizers [110, 52, 233]. The third approach relies on overparameterization, e.g., sufficiently overparameterized networks can learn the ground truth with a small generalization error using SGD from random initialization [128, 4, 8, 30]. There are also other approaches, such as compression [9, 17, 241, 41], Fourier analysis [179, 226], "double descent" risk curve [19], PAC-Bayesian framework [161, 152], and information bottleneck [194, 191].

However, most theoretical bounds fail to explain the performance of neural networks in practice [160, 9]. To get non-vacuous and tight enough bounds to be practically meaningful, some problem-specific factors should be taken into consideration, such as the low complexity (i.e., data-dependent analysis) [60, 108], or properties of the trained neural networks [198, 9, 221]. In this study, to achieve a practically meaningful bound, our analysis relies on the data distribution and the smoothness of the trained neural network. The analysis proposed in this study provides guarantees on the generalization error, and theoretical insights to guide the practical application.

As shown in Fig. 3.1, the optimization error is correlated with the loss value (for notation simplicity, the term "loss" indicates "empirical loss"), while the approximation error depends on the network size. In addition, a small loss requires a sufficient approximation ability, i.e., a large network size, which in turn leads to a small approximation error. If we assume a sufficiently small loss, which usually holds in practice, then the expected error mainly depends on the generalization error. Hence, we study the expected error/accuracy directly. In particular, we propose a mathematical framework to analyze the expected accuracy of neural

networks for classification problems. We introduce the concepts of *total cover (TC)*, *self cover (SC)*, *mutual cover (MC)* and *cover difference (CD)* to represent the data distribution, and then we use the concept of *cover complexity (CC)* as a measure of the complexity of classification problems. On the other hand, the smoothness of a neural network f is characterized by the *inverse of the modulus of continuity* δ_f. Because computing δ_f is not tractable in general, we propose an estimation using the spectral norm of the weight matrices of the neural network. The main terminologies are illustrated in Fig 3.2. By combining the properties of the data distribution and the smoothness of neural networks, we derive a lower bound for the expected accuracy, i.e., an upper bound for the expected classification error.

Subsequently, we test our theoretical bounds on several data sets, including MNIST [124], CIFAR-10 [114], CIFAR-100 [114], COIL-20 [156], COIL-100 [155], and SVHN [159]. Our numerical results not only confirm our theoretical bounds, but also provide insights into the optimization process and the learnability of neural networks. In particular, we find that:

- The best accuracy that can be achieved in practice (i.e., optimized by stochastic gradient descent) by fully-connected networks is approximately linear with respect to the cover complexity of the data set.
- The trend of the expected accuracy is consistent with the smoothness of the neural network, which provides a new "early stopping" strategy by monitoring the smoothness of the neural network.
- The neural network smoothness decreases when the network depth and width increases, with the effects of depth more significant than that of width.
- The neural network smoothness is insensitive to the training dataset size, and

is bounded by a lower positive constant. This point makes our theoretical result (Theorem 3.3.5) specifically pertinent to deep neural networks.

The chapter is organized as follows. After setting up notation and terminology in Section 3.2, we present the main theoretical bounds for the accuracy based on the data distribution and the smoothness of neural networks in Section 3.3. In Section 3.4, we provide the numerical results for several data sets. In Section 3.5 we include a discussion, and in Section 3.6 we summarize our findings.

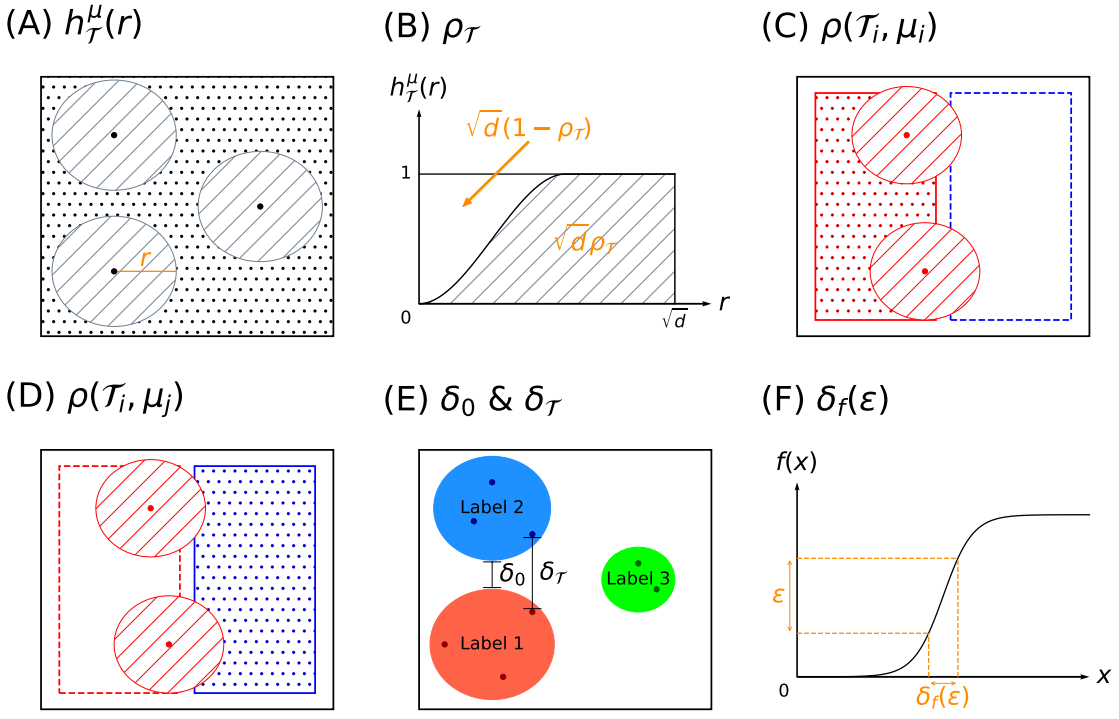


Figure 3.2: Illustrations of the main definitions and terminologies. (**A**) $h_{\mathcal{T}}^{\mu}(r)$ in Eq. (3.2): the probability of the neighborhood of the training set with radius of r. (**B**) $\rho_{\mathcal{T}}$ in Definition 3.2.5: total cover of training set $\mathcal{T}$. (**C**) $\rho(\mathcal{T}_i, \mu_i)$ in Definition 3.2.5: self cover of training set $\mathcal{T}$, which intuitively represents the aggregation of the data points of the same class. The red dots denote the probability distribution of the class "red", and the two disks are the probability of the neighbourhood of two "red" data points with respect to the distribution of the class"red" itself. (**D**) $\rho(\mathcal{T}_i, \mu_j)$ in Definition 3.2.5: mutual cover of training set $\mathcal{T}$, which intuitively represents the overlapping between the data points of two different classes. The blue dots denote the probability distribution of the class "blue", and the two disks are the probability of the neighbourhood of the two "red" data points with respect to the distribution of the class "blue". (**E**) δ_0 in Proposition 3.2.3: separation gap; $\delta_{\mathcal{T}}$ in Theorem 3.3.4: empirical separation gap. (**F**) δ_f in Definition 3.2.16: the inverse of the modulus of continuity.

3.2 Preliminaries

Before giving the main results, we introduce the necessary notation and terminology. Without loss of generality, we assume that the space we need to classify is

$$D = [0, 1]^d,$$

where d is the dimensionality, and the points in this space are classified into K categories, i.e., there are K labels $\{1, 2, \cdots, K\}$. We denote the probability measure on D by μ, i.e., for a measurable set $A \subseteq D$, $\mu(A)$ is the probability of a random sample belonging to A.

3.2.1 Ideal label function

For the problem setup, we assume that every sample has at least one true label, and one sample may have multiple true labels. Taking image classification as an example, each image has at least one correct label. A fuzzy image or an image with more than one object in it may have multiple possible correct labels, and as long as the prediction is one of these labels, we consider the prediction to be correct.

It is intuitive that when two samples are close enough, they should have similar labels, which means that the ideal label function should be continuous. The continuity of a mapping depends on the topology of both domain and image space. For the domain of the ideal label function, we choose the standard topology induced by the Euclidean metric. As for the topology of the image space, we define it as follows. We first define the label set and the topology on it.

Definition 3.2.1 (Topology). *Let*

$$T = 2^{\{1,2,\cdots,K\}} \backslash \{\varnothing\}$$

be the label set. Define the topology on T to be

$$\tau_T = \left\{ \bigcup_{k \in I} U_k \middle| I \subseteq T \right\},$$

where $U_k = \{V \in T | V \supseteq k\}$ for $k \in T$, and thus (T, τ_T) constitutes a topological space.

Analogous to the Euclidean metric topology, U_k is viewed as the open ball centered at k, and the union of some U_k is defined as the open set, see B.1 for an example. With this choice of the topology, a function $f : D \to T$ is continuous if and only if

$$\forall x \in D \forall k \in f(x) \exists \delta > 0 \; s.t. \; k \in f(y) \; if \; \|y - x\| < \delta.$$

Next we give the definition of the ideal label function according to this topological space.

Definition 3.2.2 (Ideal label function). *An idea label function is a continuous function*

$$tag : D \to T$$

where D is equipped with the Euclidean metric topology and T with the topology τ_T from Definition 3.2.1. This continuity holds if and only if

$$\forall x \in D \forall k \in tag(x) \exists \delta > 0 \; s.t. \; k \in tag(y) \; if \; \|y - x\| < \delta. \tag{3.1}$$

Eq. (3.1) means that two neighboring points would have some common labels. Based on the topological space defined above, it is easy to show that Eq. (3.1) is equivalent to continuity. The reason why we consider a multi-label setup for classification problems is that it allows for the continuity property in Eq. (3.1), which is impossible in the setup of a single label set, unless the label function is constant. In addition, the multi-label setup introduces a smooth transition, i.e., a buffer domain, between two domains of different labels, while the transition is sharp in the single label setup. In the following proposition, we show that if two samples are close enough, they must share at least one common label.

Proposition 3.2.3 (Separation gap). $\exists \delta > 0,\ s.t.\ tag(x) \cap tag(y) \neq \varnothing\ when\ \|x-y\| < \delta$. *We denote the supremum of δ as the separation gap δ_0, which is used in the sequel.*

Proof. The proof can be found in B.2. □

To understand the geometric interpretation of δ_0, we consider the following special case: the label of each sample is either a single label set, such as $\{1\}$, or the full label set $\{1, 2, \cdots, K\}$ if it is not uniquely identifiable.

Proposition 3.2.4 (Geometric interpretation of separation gap). *If the label of each sample is either a single label set or the full label set $\{1, 2, \cdots, K\}$, then δ_0 is the smallest distance between two different single label points, i.e.,*

$$\delta_0 = \inf\{\ \|x - y\| \mid tag(x) \neq tag(y), and \\ tag(x), tag(y) \in \{\{1\}, \{2\}, \cdots, \{K\}\}\ \}.$$

Proof. The proof can be found in B.3. □

3.2.2 Cover complexity of data set

In this subsection, we introduce a quantity to measure the difficulty of learning a training data set

$$\mathcal{T} = \{x_1, x_2, \ldots, x_n\} \subseteq D.$$

First, we give some notations and propositions.

With the measure μ, the probability of the neighborhood of the training set with radius of r is defined as

$$h_{\mathcal{T}}^{\mu}(r) := \mu\left(D \cap \bigcup_{x_i \in \mathcal{T}} B(x_i, r)\right), \tag{3.2}$$

where $B(x_i, r)$ is the open ball centered at x_i with radius of r, see Fig. 3.2A. Obviously, $h_{\mathcal{T}}^{\mu}(r)$ is a monotone non-decreasing function, $h_{\mathcal{T}}^{\mu}(0) = 0$, and $h_{\mathcal{T}}^{\mu}(r) = 1$ when $r > \sqrt{d}$, see Fig. 3.2B. To represent the global behavior of $h_{\mathcal{T}}^{\mu}(r)$, we use the integral of $h_{\mathcal{T}}^{\mu}(r)$ with respect to r:

$$\rho(\mathcal{T}, \mu) := \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} h_{\mathcal{T}}^{\mu}(r) dr.$$

Hence, $\rho(\mathcal{T}, \mu)$ considers both the number and location of the data points, and also the probability distribution of the space. The value $\rho(\mathcal{T}, \mu)$ is larger if the number of data points is increased and also if the probability distribution is more concentrated around $\mathcal{T}$, which we call the "coverability" of $\mathcal{T}$. We can increase $\rho(\mathcal{T}, \mu)$ by adding more data points or redistribute their locations. Next, we introduce the formal definition for the "coverability".

Definition 3.2.5 (Coverability)**.** *Let $\mathcal{T}$ be a data set from a domain D with probability*

measure μ. We define the following for the coverability of $\mathcal{T}$.

(i) The total cover (TC) is

$$\rho_{\mathcal{T}} := \rho(\mathcal{T}, \mu).$$

Thus, $0 \leq \rho_{\mathcal{T}} \leq 1$.

(ii) The cover difference (CD) is

$$CD(\mathcal{T}) := \frac{1}{K}\sum_{i} \rho(\mathcal{T}_i, \mu_i) - \frac{1}{K(K-1)}\sum_{i \neq j} \rho(\mathcal{T}_i, \mu_j),$$

where K is the number of categories, and $\mathcal{T}_i \subset \mathcal{T}$ and μ_i represent the subset and probability measure of the label i respectively, i.e.,

$$\mathcal{T}_i := \{x \in \mathcal{T} | i \in tag(x)\}, \quad \mu_i(A) := \frac{\mu(A \cap D_i)}{\mu(D_i)}$$

with $D_i := \{x \in D | i \in tag(x)\}$. Here, $\rho(\mathcal{T}_i, \mu_i)$ is called self cover (SC), and $\rho(\mathcal{T}_i, \mu_j)$ is called mutual cover (MC).

(iii) The cover complexity (CC) is

$$CC(\mathcal{T}) := \frac{1 - \rho_{\mathcal{T}}}{CD(\mathcal{T})}$$

for $CD(\mathcal{T}) \neq 0$.

Remark 3.2.6. *CD is defined as the difference between the mean of SC and the mean of MC, since each category occurs with the same probability ($\sim 1/K$) in the data sets mostly used in practice. If there are some categories occurring more frequently than others, then it is straightforward to extend this definition by using the mean weighted by the probability of each category.*

In image classification, the dimension of the image space is very high, and thus the data points are quite sparse. However, due to the fact that images actually live on a manifold of low dimension, the probability density around $\mathcal{T}$ is actually high, which makes the TC to be meaningful. In our next result, we derive a lower bound of $h^{\mu}_{\mathcal{T}}(r)$ by $\rho_{\mathcal{T}}$.

Proposition 3.2.7. *Let $\mathcal{T}$ be a data set. $h^{\mu}_{\mathcal{T}}(r)$ and $\rho_{\mathcal{T}}$ are defined as above. Then we have*

$$h^{\mu}_{\mathcal{T}}(r) \geq 1 - \frac{\sqrt{d}}{r}(1 - \rho_{\mathcal{T}}), \quad 0 < r \leq \sqrt{d}.$$

Proof. The proof can be found in B.4. □

From this proposition, we know that for a fixed r, $h^{\mu}_{\mathcal{T}}(r)$ is close to 1 if $\rho_{\mathcal{T}}$ is large enough. However, the probability distribution is usually given in practice, and we can only control the number of samples. The following theorem shows that $\rho_{\mathcal{T}}$ can be arbitrary close to 1 when enough samples are available.

Theorem 3.2.8. *Let $\mathcal{T}$ be a data set of size n drawn according to μ. Then there exists a non-increasing function $\varrho(\epsilon)$ satisfying $\lim_{\epsilon \to 0} \varrho(\epsilon) = 1$, and for any $0 < \eta, \epsilon \leq \frac{1}{2}$, there exists an*

$$m = O\left(\frac{d+1}{\epsilon} \ln \frac{d+1}{\epsilon} + \frac{1}{\epsilon} \ln \frac{1}{\eta}\right),$$

such that

$$\rho_{\mathcal{T}} \geq \varrho(\epsilon)$$

holds with probability at least $1 - \eta$ when $n \geq m$.

Proof. The proof and some other results regarding TC can be found in B.5. □

The reason why CD is introduced is that TC does not consider the labels of the data points. However, data points of the same label should be clustered in a good data set. $CD(\mathcal{T})$ is the difference of self cover and mutual cover, which considers the distributions of each label. By normalizing TC with CD, the cover complexity $CC(\mathcal{T})$ is able to measure the difficulty of learning a data set. The difficulty of a problem should be translation-independent and scale-independent. It is easy to see that $CC(\mathcal{T})$ is independence of translation, and the following proposition shows that it is also scale-independent.

Proposition 3.2.9 (Scale independence)**.** *$CC(\mathcal{T})$ is scale-independent, i.e., if all the data points are scaled by the same factor less than 1, then $CC(\mathcal{T})$ is unchanged.*

Proof. The proof can be found in B.6. □

3.2.3 Setup for accuracy analysis

The setup for accuracy analysis is as follows.

Definition 3.2.10. *If $f : D \to \mathbb{R}^K$ is a continuous mapping, then the mapping*

$$\hat{f} : x \mapsto \frac{e^{f(x)}}{\sum_i e^{f_i(x)}}$$

is still continuous, where $f_i(x)$ represents the i-th component of $f(x)$, and $e^{f(x)} \in \mathbb{R}^K$ is the componentwise exponent of $f(x)$. We have $\hat{f}_i(x) > 0$, and $\sum_i \hat{f}_i(x) = 1$. For convenience, we directly consider the case that $f_i(x) > 0$ and $\sum f_i(x) = 1$, and we call such mapping the normalized continuous positive mapping.

Remark 3.2.11. *A neural network with softmax nonlinearity is a normalized continuous positive mapping.*

Different from the accuracy usually used in classification problems, we define a stronger accuracy called c-accuracy as follows.

Definition 3.2.12 (c-accuracy at x)**.** *Let f be a normalized continuous positive mapping. For $0.5 \leq c < 1$, we say that f is c-accurate at point x if*

$$\exists i_{max} \in tag(x),\ s.t.\ f_{i_{max}}(x) > c.$$

Definition 3.2.13 (c-accuracy on D)**.** *Let f be a normalized continuous positive mapping. The c-accuracy of f on a sample space D is defined as*

$$p_c(f) := \frac{\mu(H_c^f)}{\mu(D)} = \mu(H_c^f),$$

where $H_c^f := \{x \in D | f$ is c-accurate at $x\}$.

Definition 3.2.14 (c-accuracy on $\mathcal{T}$)**.** *Let f be a normalized continuous positive mapping. The c-accuracy of f on a data set $\mathcal{T}$ is defined as*

$$p_c^{\mathcal{T}} := \frac{\rho_{\widetilde{\mathcal{T}}}}{\rho_{\mathcal{T}}},$$

where $\widetilde{\mathcal{T}} := \{x \in \mathcal{T} | f$ is c-accurate at $x\}$, and $\rho_{\widetilde{\mathcal{T}}}$ and $\rho_{\mathcal{T}}$ are the TC of $\widetilde{\mathcal{T}}$ and $\mathcal{T}$, respectively.

Definition 3.2.15 (Expected accuracy)**.** *Let f be a normalized continuous positive mapping. The expected accuracy of f on a sample space D is defined as*

$$p(f) := \frac{\mu(H^f)}{\mu(D)} = \mu(H^f),$$

where $H^f := \{x \in D | \exists i_{max} \in tag(x),\ s.t.\ f_{i_{max}}(x) > f_i(x)\ \forall 1 \leq i \leq K,\ i \neq i_{max}\}$.

We note that the c-accuracy of f on D represents the expected c-accuracy, and

the c-accuracy of f on $\mathcal{T}$ represents the empirical c-accuracy.

Finally, we define a non-decreasing function δ_f to describe the smoothness of f.

Definition 3.2.16 (Smoothness)**.** *Let $f : D \to \mathbb{R}^K$ be a continuous mapping. Then f is uniformly continuous due to the compactness of D, i.e.*

$$\forall \epsilon > 0,\ \exists \delta > 0,\ s.t.\ \|f(x) - f(y)\|_\infty < \epsilon\ when\ \|x - y\| < \delta.$$

We denote the supremum of δ satisfying the above requirement by $\delta_f(\epsilon)$. It is easy to see that $\delta_f(\epsilon)$ is equal to the inverse of the modulus of continuity of f.

For low dimensional problems, we can directly compute δ_f by brute force. However, for high dimensional problems, it is intractable to compute δ_f, and thus we give the following lower bound of δ_f for a fully-connected neural network f with softmax, which is also the main network structure considered through this work.

A fully-connected neural network is defined as follows:

$$\phi_i(x) = \sigma(W_i x + b_i), 1 \leq i \leq l - 1,$$

$$f(x) = softmax(W_l(\phi_{l-1} \circ \cdots \circ \phi_1(x)) + b_l),$$

$$W_i \in \mathbb{R}^{n_i \times n_{i-1}}, b_i \in \mathbb{R}^{n_i}, 1 \leq i \leq l,$$

where n_i is the number of neurons in the layer i ($n_0 = d$ and $n_l = K$), and σ is the

activation function. Then for the ReLU activation function, we have

$$\delta_f(\epsilon) \geq \frac{\epsilon}{Lip(f)} \geq \frac{\epsilon}{\|W_1\|_2 \cdots \|W_l\|_2 \cdot Lip(softmax)}, \tag{3.3}$$

where $\|\cdot\|_2$ is the spectral norm, $Lip(f)$ and $Lip(softmax)$ represent the Lipschitz coefficients of f and $softmax$, respectively. $Lip(softmax)$ is a constant less than $1/2$, and thus is ignored in our numerical examples. We note that although the lower bound of $\delta_f(\epsilon)$ depends exponentially on the neural net depth, $\delta_f(\epsilon)$ itself does not necessarily scale exponentially with the network depth.

3.3 Lower bounds for the expected accuracy

In this section, we present a theoretical analysis of the lower bound for the expected accuracy as well as an upper bound for the expected error.

Proposition 3.3.1. *Let f be a normalized continuous positive mapping. Suppose that $\mathcal{T}$ is a single label training set, i.e. $tag(\mathcal{T}) \subseteq \{\{1\}, \{2\}, \cdots, \{K\}\}$. For any $0.5 \leq c_2 < c_1 \leq 1$, we have*

$$p_{c_2}(f) \geq 1 - \frac{\sqrt{d}}{\delta}(1 - p_{c_1}^{\mathcal{T}}\rho_{\mathcal{T}}),$$

where $\delta = \min(\delta_0, \delta_f(c_1 - c_2))$.

Proof. The proof can be found in B.7. □

Proposition 3.3.1 shows that the expected c_2-accuracy of f can be bounded by the empirical c_1-accuracy and the TC of the training set. We can see that $p_{c_2}(f)$ tends to 1 when $\rho_{\mathcal{T}}$ and $p_{c_1}^{\mathcal{T}}$ tend to 1. Next we derive a bound for the accuracy by

taking into account the loss function.

Theorem 3.3.2 (Lower bound of c-accuracy)**.** *Let f be a normalized continuous positive mapping. Suppose that $\mathcal{T} = \{x_1, \cdots, x_n\}$ is a single label training set, and $tag(x_i) = \{k_i\}$. For any $0.5 \leq c < 1$, if the maximum cross entropy loss*

$$L_f^{max} := \max_{1 \leq i \leq n} \ell(f(x_i), k_i) = -\min_{1 \leq i \leq n} \ln(f_{k_i}(x_i)) < -\ln c,$$

then we have

$$p_c(f) \geq 1 - \frac{\sqrt{d}}{\delta}(1 - \rho_{\mathcal{T}}),$$

where ℓ is the cross entropy loss that $\ell(f(x), k) = -\ln(f_k(x))$, $\delta = \min(\delta_0, \delta_f(e^{-L_f^{max}} - c))$, and δ_0 is defined in Proposition 3.2.3.

Proof. The proof can be found in B.8. □

Corollary 3.3.3. *Let f be a normalized continuous positive mapping. Suppose that $\mathcal{T} = \{x_1, \cdots, x_n\}$ is a single label training set, and $tag(x_i) = \{k_i\}$. For any $0.5 \leq c < 1$, if the loss function*

$$L_f := \frac{1}{n} \sum_{1 \leq i \leq n} \ell(f(x_i), k_i) = -\frac{1}{n} \sum_{1 \leq i \leq n} \ln(f_{k_i}(x_i)) < -\frac{1}{n} \ln c,$$

then we have

$$p_c(f) \geq 1 - \frac{\sqrt{d}}{\delta}(1 - \rho_{\mathcal{T}}),$$

where ℓ is the cross entropy loss, and $\delta = \min(\delta_0, \delta_f(e^{-nL_f} - c))$.

Proof. The corollary can be obtained from Theorem 3.3.2 based on the fact $L_f^{max} \leq nL_f < -\ln c$. □

Theorem 3.3.2 reveals that the expected accuracy is related to the total cover $\rho_{\mathcal{T}}$, separation gap δ_0, neural network smoothness δ_f, and loss value L_f^{max}. We will show numerically in Section 3.4 that $\delta_f(e^{-L_f^{max}} - c)$ increases first and then decreases during the training of neural networks. The following theorem states that the maximum value of $\delta_f(e^{-L_f^{max}} - c)$ is bounded by the empirical separation gap.

Theorem 3.3.4 (Empirical separation gap)**.** *Let f be a normalized continuous positive mapping. Suppose that $\mathcal{T} = \{x_1, \cdots, x_n\}$ is a single label training set. For any $0.5 \leq c < 1$, if $L_f^{max} < -\ln c$, then we have*

$$\delta_f(e^{-L_f^{max}} - c) \leq \delta_{\mathcal{T}}/2,$$

where

$$\delta_{\mathcal{T}} := \min_{tag(x_i) \neq tag(x_j)} \|x_i - x_j\| \geq \delta_0$$

is called the empirical separation gap, i.e., the smallest distance between two differently labeled training points.

Proof. The proof can be found in B.9. □

Besides the upper bound, the lower bound of δ_f is also important to the accuracy. We have observed that in practice the low bound of δ_f exists (Figs. 3.5, 3.6, 3.7 and 3.8), which indicates the existence of κ in the following theorem. Based on this observation, we have the following theorem for the accuracy.

Theorem 3.3.5 (Lower bound of accuracy)**.** *Assume that there exists a constant $\kappa > 0$, such that*

$$\kappa\delta_0 \leq \kappa\delta_{\mathcal{T}} \leq \delta_f(e^{-L_f^{max}} - 0.5) \leq \delta_{\mathcal{T}}/2 \tag{3.4}$$

holds for any single label training set $\mathcal{T}$ and any trained network f on $\mathcal{T}$ such that $L_f^{max}(\mathcal{T}) < \ln 2$, then we have the following conclusions for the expected accuracy $p(f)$ and the expected error $\mathcal{E}(f) = 1 - p(f)$:

(i) *with the same condition of Theorem 3.3.2,*

$$p(f) \geq 1 - \frac{\sqrt{d}}{\delta}(1 - \rho_{\mathcal{T}}),$$

(ii) $$\mathcal{E}(f) \leq \alpha(\mathcal{T}) \cdot CC(\mathcal{T}),$$

(iii) $$\lim_{\rho_{\mathcal{T}} \to 1} p(f) = 1,$$

where $\delta = \min(\delta_0, \delta_f(e^{-L_f^{max}} - c))$, *and* $\alpha(\mathcal{T}) = \frac{\sqrt{d} \cdot CD(\mathcal{T})}{\min(\delta_0, \kappa\delta_{\mathcal{T}})}$.

Proof. (i) is the conclusion of Theorem 3.3.2. The proof of (ii) and (iii) can be found in B.10. □

Here, the cover complexity $CC(\mathcal{T})$ consists of two parts, one represents the richness of the whole training set while the other part describes the degree of separation between different labeled subsets. As for $\alpha(\mathcal{T})$, both the denominator and numerator seem to have a positive correlation with respect to separation level. What we wish is that $\alpha(\mathcal{T})$ is almost close to a constant with high probability and the expected error $\mathcal{E}(f)$ is mainly determined by $CC(\mathcal{T})$, which approximately represents the complexity level of the data set. We will provide more information in detail in the section concerning the numerical results.

3.4 Numerical results

In this section, we use numerical simulations to test the accuracy of neural networks in terms of the data distribution (cover complexity), and neural network smoothness. In addition, we study the effects of the network size and training dataset size on the smoothness. The codes are published in GitHub (https://github.com/jpzxshi/generalization).

3.4.1 Data distribution

Table 3.1: Cover complexity $CC(\mathcal{T})$, best error $\mathcal{E}(f)$, and normalized error $\frac{\mathcal{E}(f)}{\sqrt{K}}$ of different data sets. Different variants of data sets are used, including the original RGB or grey images (Original), grey images (Grey), and images extracted from a CNN (Conv). Images in CIFAR-100 have two variants: 100 categories (fine) and 20 categories (coarse). See B.11 for details.

Data Set	Variants	Input dim (d)	Output dim (K)	$\rho_{\mathcal{T}}$	$CD(\mathcal{T})$	$CC(\mathcal{T})$	$\mathcal{E}(f)$	$\frac{\mathcal{E}(f)}{\sqrt{K}}$
MNIST	Original	784	10	.8480	.1053	1.442	.01	.0032
CIFAR-10	Original	3072	10	.8332	.0163	10.23	.45	.1423
CIFAR-10	Grey	1024	10	.8486	.0125	12.11	.53	.1676
CIFAR-10	Conv	1024	10	.9505	.0094	5.280	.18	.0569
SVHN	Original	3072	10	.9034	.0076	12.68	.49	.1550
SVHN	Grey	1024	10	.9117	.0084	10.48	.56	.1771
SVHN	Conv	1024	10	.9632	.0123	2.995	.23	.0727
CIFAR-100	Original (coarse)	3072	20	.8337	.0185	9.012	.62	.1386
CIFAR-100	Grey (coarse)	1024	20	.8541	.0132	11.08	.72	.1610
CIFAR-100	Conv (coarse)	1024	20	.9626	.0070	5.326	.40	.0894
COIL-20	Original	16384	20	.9176	.2385	.3453	.03	.0067
CIFAR-100	Original (fine)	3072	100	.8337	.0270	6.149	.73	.0730
CIFAR-100	Grey (fine)	1024	100	.8541	.0198	7.380	.81	.0810
CIFAR-100	Conv (fine)	1024	100	.9457	.0136	4.000	.52	.0520
COIL-100	Original	49152	100	.9430	.1944	.2930	.01	.0010

In this subsection, we explore how $CC(\mathcal{T})$ affects the expected error $\mathcal{E}(f)$. In our experiments, we test several data sets, including MNIST [124], CIFAR-10 [114], CIFAR-100 [114], COIL-20 [156], COIL-100 [155], SVHN [159]. In addition to the original data set, we also create some variants: (1) the images of grey color, (2) the images extracted from a convolutional layer after training the original data set using a convolutional neural network (CNN), (3) combine several categories into

one category to reduce the number of total categories, see Table 3.1 and details in B.11.

For a training data set $\mathcal{T}$, we estimate $h^{\mu}_{\mathcal{T}}(r)$ by the proportion of the test data points within the balls with radius r centered at training data points, i.e.,

$$h^{\mu}_{\mathcal{T}}(r) \approx \frac{\text{\# Test points within radius-}r\text{ balls of training points}}{\text{\# Test data points}},$$

and then $\rho_{\mathcal{T}}$ is obtained by Definition 3.2.5. Similarly, we estimate $CD(\mathcal{T})$ and then compute $CC(\mathcal{T})$. Next for each data set, we train fully-connected neural networks with different hyperparameters, and record the best error we observed, see the details in B.11. The cover complexity and the best error for each data set are shown in Table 3.1.

These data sets are divided into three groups according to their output dimensions. For each group of the same output dimension, the error is almost linearly correlated with $CC(\mathcal{T})$, see Fig. 3.3A, regardless of the input dimension. In addition, we find that all the cases collapse into a single line when normalizing the error $\mathcal{E}(f)$ by a factor of $1/\sqrt{K}$, see Fig. 3.3B.

It is noteworthy that the $CC(\mathcal{T})$ of convolutional variants of data sets is much smaller than the original data sets, and hence the expected accuracy increases. The results confirm the importance of data distribution.

Next, we consider the most difficult data set, i.e., data with random labels. We choose MNIST and then assign each image a random label. We repeat this process 50 times, and compute each $CC(\mathcal{T})$. The distribution of $|CC(\mathcal{T})|$ is shown in Fig. 3.4. The smallest $|CC(\mathcal{T})|$ is ~300, which is much larger than that of the

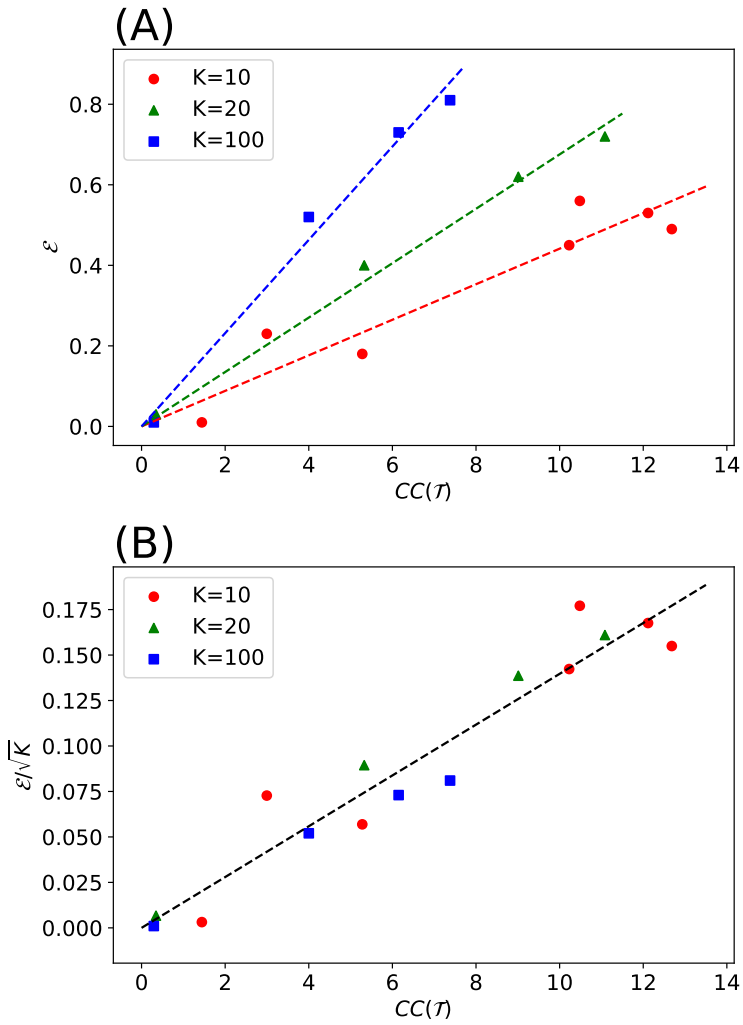


Figure 3.3: Relationship between cover complexity and error achieved by fully-connected neural networks. (**A**) Linear relationship between cover complexity $CC(\mathcal{T})$ and error $\mathcal{E}(f)$ for different data sets with different category number K. The coefficients of determination of the linear regression (with the constraint that the line must pass through the origin) are $R^2 \approx 0.89, 0.99, 0.98$ for $K = 10, 20, 100$, respectively. (**B**) Linear relationship between cover complexity $CC(\mathcal{T})$ and normalized error $\frac{\mathcal{E}(f)}{\sqrt{K}}$ for different data sets. Red, green and blue points represent data sets with output dimension of 10, 20 and 100, respectively. The line is fitted as $\frac{\mathcal{E}(f)}{\sqrt{K}} \approx 0.014 CC(\mathcal{T})$, and the coefficient of determination is $R^2 \approx 0.92$.

original data sets with $CC(\mathcal{T}) < 20$. This extreme example again confirms that $CC(\mathcal{T})$ is a proper measure of the difficulty of classifying a data set.

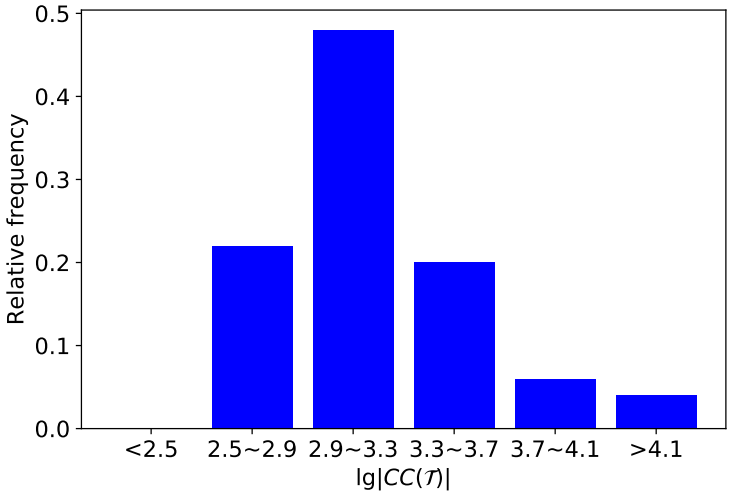


Figure 3.4: Distribution of $CC(\mathcal{T})$ of labeled MNIST data set. We assign each image in MNIST a random label and compute its $CC(\mathcal{T})$. The smallest $|CC(\mathcal{T})|$ is ~300. The distribution is obtained from 50 random data sets.

3.4.2 Neural network smoothness

In this subsection, we will investigate the relationship between the neural network smoothness $\delta_f(e^{-L_f^{max}} - c)$ and the accuracy, and the effects of network size (depth and width) on the smoothness. We first show results for one- and two-dimensional problems, where $\delta_f(e^{-L_f^{max}} - c)$ can be computed accurately by brute force. Subsequently, we consider the high dimensional setting of the MNIST data set, where we estimate $\delta_f(e^{-L_f^{max}} - c)$ by Eq. (3.3).

One- and two-dimensional problems

We first consider a one-dimensional case and a two-dimensional case. For the one-dimensional case, we choose the sample space $D = [0, 1]$, $K = 2$, and the ideal label function as

$$tag(x) = \begin{cases} \{1\} & \text{if } x \in [0, 0.5 - \frac{\delta_0}{2}] \\ \{1, 2\} & \text{if } x \in (0.5 - \frac{\delta_0}{2}, 0.5 + \frac{\delta_0}{2}) \\ \{2\} & \text{if } x \in [0.5 + \frac{\delta_0}{2}, 1] \end{cases},$$

with separation gap $\delta_0 = 0.1$. We use n equispaced points ($n \geq 4$ is an even number) on $D \setminus (0.5 - \frac{\delta_0}{2}, 0.5 + \frac{\delta_0}{2})$ as the training set, i.e., $\mathcal{T} = \mathcal{T}_1 \cup \mathcal{T}_2$, where

$$\mathcal{T}_1 = \left\{0, \frac{1 - \delta_0}{n - 2}, \frac{1 - \delta_0}{n - 2} \cdot 2, \ldots, \frac{1}{2} - \frac{\delta_0}{2}\right\},$$
$$\mathcal{T}_2 = \left\{\frac{1}{2} + \frac{\delta_0}{2}, \ldots, 1 - \frac{1 - \delta_0}{n - 2}, 1\right\}.$$

We choose 10000 equispaced points on D as the test data.

For the two-dimensional case, we choose the sample space $D = [0, 1]^2$, $K = 2$, and the ideal label function as

$$tag(\mathbf{x}) = \begin{cases} \{1\} & \text{if } \|\mathbf{x} - (0.5, 0.5)\| \leq 0.4 - \frac{\delta_0}{2} \\ \{2\} & \text{if } \|\mathbf{x} - (0.5, 0.5)\| \geq 0.4 + \frac{\delta_0}{2} \\ \{1, 2\} & \text{otherwise} \end{cases},$$

with $\delta_0 = 0.1$. For the training set, we first choose $n = m^2$ equispaced points, i.e., $\mathcal{T} = \{0, \frac{1}{m-1}, \frac{2}{m-1}, \ldots, \frac{m-2}{m-1}, 1\}^2$, and then remove the points with label $\{1, 2\}$ to ensure that all samples are of single label. We choose 1000^2 equispaced points on

D as the test data.

In our experiments, we use a 3-layer fully-connected NN with ReLU activation and 30 neurons per layer. The neural network is trained for 1000 iterations by the Adam optimizer [112] for the one-dimensional problem, and 2000 iterations for the two-dimensional problem. For the one-dimensional problem, the c-accuracy $p_c(f)$ with $c = 0.5$ and lower bounds for different numbers of training points are listed in Table 3.2. We can see that the bounds become tighter when n is larger.

Table 3.2: Comparison between c-accuracy $p_c(f)$ with $c = 0.5$ and the lower bounds for different training set sizes for the one-dimensional problem. Here δ is indicated in Theorem 3.3.2. The neural network is trained for 1000 iterations by the Adam optimizer.

n	L_f^{max}	$\rho_{\mathcal{T}}$	δ	$p_c(f)$	$h_{\mathcal{T}}(\delta)$	$1-\frac{\sqrt{d}}{\delta}(1-\rho_{\mathcal{T}})$
10	.285	.972	.045	1.0	0.80	0.38
20	.246	.988	.041	1.0	1.00	0.69
40	.182	.994	.041	1.0	1.00	0.85
80	.127	.997	.038	1.0	1.00	0.92

During the training process of the neural network, the test loss first decreases and then increases, while δ_f first increases and then decreases, see Fig. 3.5A for the one-dimensional problem ($n = 20$) and Fig. 3.5B for the two-dimensional problem ($n = 400$). δ_f is bounded by $\delta_{\mathcal{T}}/2$, as proved in Theorem 3.3.4. We also observe that the trends of test loss and δ_f coincide, and thus we should stop the training when δ_f begins to decrease to prevent overfitting.

High-dimensional problem

In the high-dimensional problem of MNIST, we consider the average loss L_f instead of the maximum loss L_f^{max}, which is very sensitive to extreme points. As shown

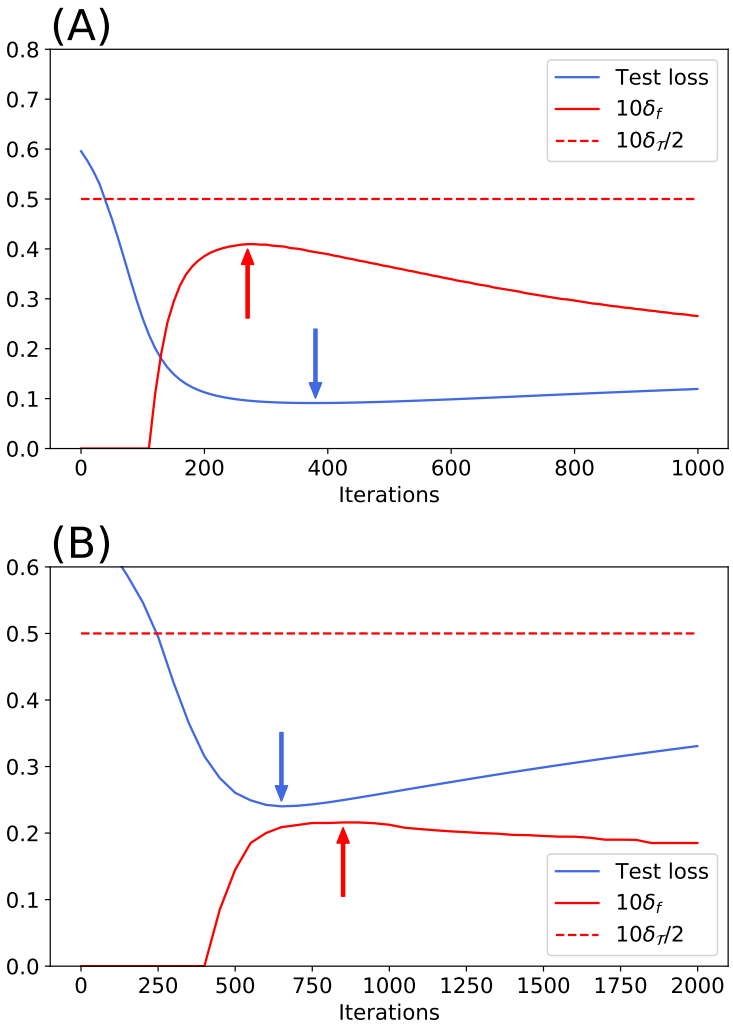


Figure 3.5: Consistency between the test loss and neural network smoothness $\delta_f(e^{-L_f^{max}} - \frac{1}{2})$ during the training process of the neural network. (**A**) One-dimensional problem of $n = 20$. (**B**) Two-dimensional problem of $n = 400$. δ_f is bounded by $\delta_{\mathcal{T}}/2$. The arrows indicate the minimum of the test loss and the maximum of δ_f.

in Eq. (3.3), we use the following quantity to bound $\delta_f(e^{-L_f} - c)$:

$$\Delta_f = \frac{e^{-L_f} - c}{\|W_1\|_2 \cdots \|W_l\|_2}.$$

Because we use the c-accuracy to approximate the true accuracy, for the classification problems with two categories, $p_{0.5}(f)$ and $p(f)$ are equivalent. However, they are not equal for problems with more than two categories, where the best c depends on the properties of the data set, such as the easiness of learning to classify the data set. If the data set is easy to classify, such as MNIST, the best c should be close to 1. In our example, we choose $c = 0.9$. We train MNIST using a 3-layer fully-connected NN with ReLU activation and 100 neurons per layer for 100 epochs. In Fig. 3.6, we can also see the consistency between the test loss and neural network smoothness, as we observed in the low-dimensional problems.

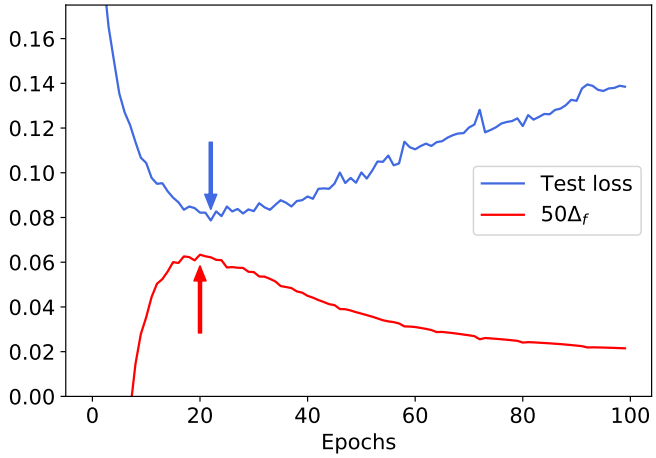


Figure 3.6: Consistency between test loss and neural network smoothness during the training of the neural network for MNIST. The arrows indicate the minimum of the test loss and the maximum of Δ_f.

Effects of the network size and training dataset size on the smoothness

We have demonstrated that network smoothness $\delta_f(e^{-L_f^{max}} - c)$ is an important factor to the accuracy. Next, we investigate the effects of network size (depth and width) on the smoothness for binary classification problems, which are explained as follows. We consider the one-dimensional sample space $D = [0, 1]$, and choose n equispaced points on D as the training data locations. To avoid the effects of the choice of target true functions, we always repeat experiments with different target functions, and in each experiment we generate a random target function. Specifically, to generate a random target function, we first sample two random functions $g_1(x)$ and $g_2(x)$ from a Gaussian process with the radial basis function kernel of a length scale 0.2, and then assign a point x as category 1 if $g_1(x) > g_2(x)$, otherwise assign this point as category 2. When training neural networks, we monitor the value of $\delta_f(e^{-L_f^{max}} - c)$ and stop the training once δ_f begins to decrease as shown in Figs. 3.5 and 3.6. We first choose the dataset size $n = 10$, and we show that the normalized smoothness δ_f/δ_T decreases as the network depth or width increases (Fig. 3.7). We also show that the effects of depth is more significant than that of width.

Our main theorem (Theorem 3.3.5) requires the assumption (Eq. 3.4), which would not be true for any machine learning algorithms. Here, we verify this assumption for neural networks by numerical experiments. Specifically, we train a fully-connected neural networks using training datasets of different size n. We show that $\delta_f/\delta_{\mathcal{T}}$ is insensitive to training dataset size, and is always bounded by a lower positive constant κ (Fig. 3.8). This result reveals that the neural networks would fit a dataset in a relatively smooth way during the training process.

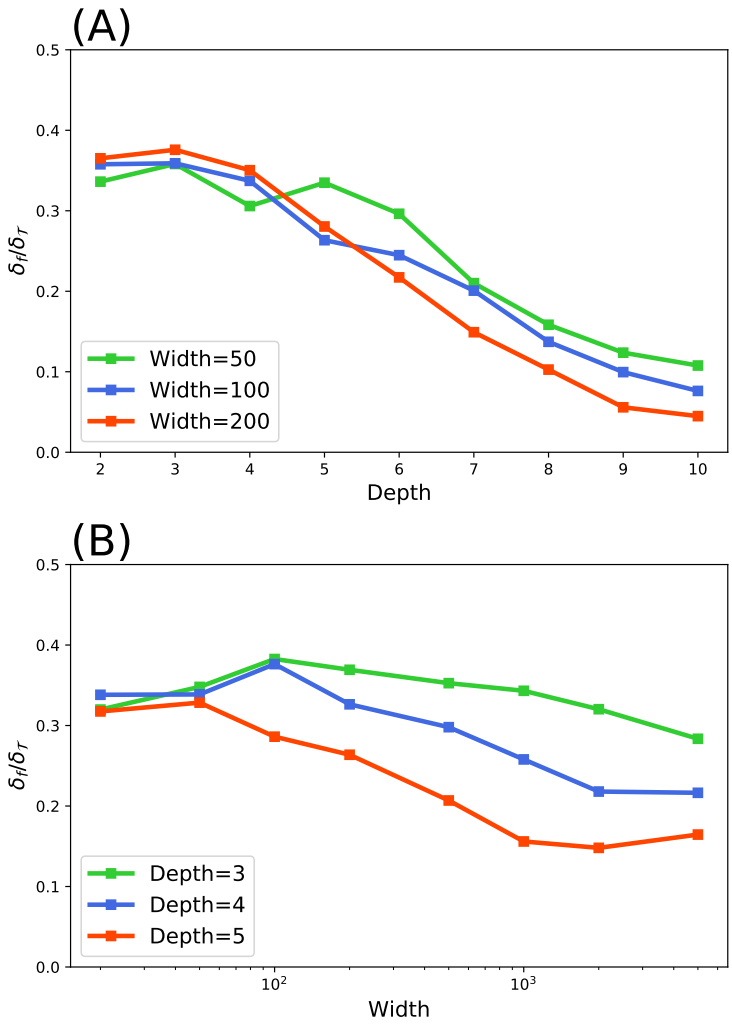


Figure 3.7: Effects of network depth and width on the normalized smoothness $\delta_f/\delta_\mathcal{T}$. (**A**) The $\delta_f/\delta_\mathcal{T}$ decreases fast when the network depth increases. (**B**) The $\delta_f/\delta_\mathcal{T}$ decreases relatively slow when the network width increases. The results are averaged from 50 independent experiments.

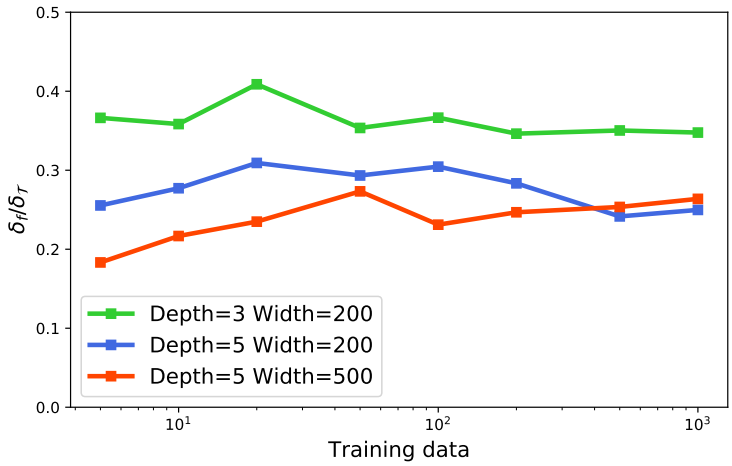


Figure 3.8: Effects of the number of the training data on the normalized $\delta_f/\delta_\mathcal{T}$. The $\delta_f/\delta_\mathcal{T}$ is insensitive to the number of the training data, and is bounded by a lower positive constant. The results are averaged from 50 independent experiments.

3.5 Discussion

When neural networks are used to solve classification problems, we expect that the accuracy is dependent on some properties of the data set. It is still quite surprising, however, that there is a linear relationship between the accuracy and the cover complexity of the data set, as we have seen in Section 3.4.1. Theorem 3.3.5(ii) provides an upper bound of the error, but a lower bound is missing. To fully explain this observation, two conjectures of the learnability of fully-connected neural networks are proposed: when a neural network f is trained on a data set $\mathcal{T}$ in such a way that $L_f^{max}(\mathcal{T}) < \ln 2$ and $\delta_f(e^{-L} - c) \asymp \delta_\mathcal{T}$, then we have

- $\mathcal{E}(f) \approx c(K) \cdot CC(\mathcal{T})$, where $c(K)$ is a constant depending only on K.
- $\frac{\mathcal{E}(f)}{\sqrt{K}} \approx c \cdot CC(\mathcal{T})$, where $c \approx 0.014$ is a constant.

On the other hand, the theoretical and numerical results provide a better understanding of the generalization of neural network from the training procedure.

The smoothness $\delta_f(e^{-L} - c)$ of neural networks plays a key role, where L is the maximum loss L_f^{max} or the average loss L_f. We can see that:

- $\delta_f(e^{-L} - c)$ depends on both the regularity of f and the loss value L (which also depends on f). Large $\delta_f(e^{-L} - c)$ requires good regularity and large $e^{-L} - c$, i.e., small L. However, small L could correspond to bad regularity of f. Thus, there is a trade-off between the loss value L and the regularity of f.
- Due to this trade-off, $\delta_f(e^{-L} - c)$ increases first and then decrease during the training process. Hence, we should not optimize neural networks excessively. Instead, we should stop the training early when $\delta_f(e^{-L} - c)$ begins to decrease, which leads to another "early stopping" strategy to prevent overfitting.

We also note that the lower bound of $\delta_f(e^{-L} - c)$ in Eq. (3.3) relates to the norm of weight matrices of neural networks:

$$\delta_f(e^{-L} - c) \gtrsim \frac{e^{-L} - c}{\|W_1\|_2 \cdots \|W_l\|_2}.$$

There have been some works to study the norm-based complexity of neural networks (see the Introduction), and these bounds typically scale with the product of the norms of the weight matrices, e.g., [160]

$$\frac{1}{\gamma_{margin}^2} \prod_{i=1}^{l} h_i \|W_i\|_2^2,$$

where h_i and W_i are the number of nodes and the weight matrix in layer i of a network with l-layers, and γ_{margin} is the margin quantity, which describes the goodness of fit of the trained network to the data. The product of the matrix norms depends exponentially on the depth, while some recent works show that

the generalization bound could scale polynomially in depth under some assumptions [152, 221]. The exploration of the dependence of $\delta_f(e^{-L} - c)$ on depth is left for future work.

3.6 Conclusion

In this chapter, we study the generalization error of neural networks for classification problems in terms of the data distribution and neural network smoothness. We first establish a new framework for classification problems. We introduce the *cover complexity* (CC) to measure the difficulty of learning a data set, an accuracy measure called *c-accuracy* which is stronger than the standard classification accuracy, and *the inverse of the modulus of continuity* to quantify neural network smoothness. Subsequently, we derive a quantitative bound for the expected accuracy/error in Theorem 3.3.5, which considers both the cover complexity and neural network smoothness.

We validate our theoretical results by several data sets of images. Our numerical results demonstrate that CC is a reliable measure for the difficulty of learning to classify a data set. On the other hand, we observe a clear consistency between test loss and neural network smoothness during the training process. We also show that neural network smoothness decreases when the network depth and width increases, and the effects of depth is more significant than that of width, while the smoothness is insensitive to training dataset size.

Chapter

FOUR

Extraction of Mechanical Properties of Materials through Deep Learning from Instrumented Indentation

Manuscript information

A version of this chapter appeared in [134]: L. Lu*, M. Dao*, P. Kumar, U. Ramamurty, G. E. Karniadakis, & S. Suresh. Extraction of mechanical properties of materials through deep learning from instrumented indentation. *Proceedings of the National Academy of Sciences*, 117(13), 7052–7062, 2020.

Author contributions: L.L., M.D., U.R., G.E.K., and S.S. designed research. L.L., M.D., and P.K. performed research. L.L., M.D., P.K., U.R., G.E.K., and S.S. analyzed data. L.L., M.D., and G.E.K. developed the multifidelity deep-learning algorithms. G.E.K. and S.S. supervised the project. L.L., M.D., P.K., U.R., G.E.K., and S.S. wrote the paper.

4.1 Introduction

4.1.1 Instrumented indentation for extracting mechanical properties of materials

Instrumented indentation has been a research topic for scientific investigations as well as industrial applications during the past several decades [168, 207, 42, 70, 50, 46, 169, 74, 121]. Here, the loading force (P) of the indenter tip and the resultant depth of penetration (h) of the tip into the material are continuously recorded both during loading and unloading. Such depth-sensing or instrumented indentation has, in recent years, emerged as an appealing means of probing the mechanical properties of hard and soft materials, devices, components and structures over

a wide spectrum of size scales, from nanometers to meters, with sufficient resolution to measure forces over the range of micronewtons to kilonewtons, and displacements over the range of nanometers to centimeters [169]. Some key advantages of instrumented indentation as a method to extract properties include the need to test only a relatively small volume of the material (in relation to its overall volume), which for many applications would render it an essentially nondestructive probe. Furthermore, it offers the potential to determine processing-induced residual stresses [207], anisotropic properties [104], property gradients arising from compositional, microstructural or residual-stress gradients in materials [206, 44], as well as coupled electrical-mechanical responses of integrated systems such as those involving piezoelectric materials [201, 188]. It is especially suited for extracting material properties in a wide variety of applications involving additive manufacturing, near-net-shape manufacturing, and integrated manufacturing (e.g., load-bearing mechanical structures and components embedded with electronic, optical, magnetic or biological components). Indeed, in some cases it may be the only viable and practical method for determining local and volume-averaged properties, as, for example, in the context of evaluating in-situ thin-film mechanical properties or mapping the local mechanical property variations across grain/phase boundaries or along gradients in structures [74]. Similarly, in order to evaluate detailed layer-by-layer mechanical characteristics of a three-dimensionally (3D) printed material, instrumented indentation appears to be the only practically viable method to probe how processing conditions lead to evolution of properties and structural integrity. With the commercial availability of sophisticated and inexpensive robotic tools, depth-sensing indentation measurements can readily be incorporated into the processing and fabrication of materials and components by such means as layer-by-layer additive manufacturing.

In general, a forward indentation algorithm for a metallic material provides indentation response (i.e., force vs displacement [P vs. h] curve during indentation loading and retraction of the indenter) for a given set of elastoplastic properties (elastic modulus, Poisson's ratio, yield strength, strain-hardening exponent and tensile strength). The inverse indentation problem, on the other hand, should lead to the unique determination of the elastoplastic properties from a set of indentation $P - h$ data.

The hardness (H) of a material has long been used as a property from which yield strength (σ_y) can be estimated [169, 74, 210, 236], although their connection is known to be only highly approximate [50, 236]. In order to address this limitation of simple hardness measures, dimensional analyses and scaling functions have been developed [50, 74, 43] and explicit universal scaling functions have been established for solving the forward and inverse problems in depth-sensing indentation by recourse to single [50] and multiple sharp indenter tip geometries [46, 28, 31, 220]. Additionally, studies have also focused on efforts to extract elastoplastic properties from load-displacement curves for spherical indenters [139, 32, 166, 122, 129]. Due to the inherent difficulties in accurately accounting for the contact area and the initial contact depth involved with spherical indentation, and given that spherical indentation introduces an additional dimension (i.e., the indenter diameter), which needs to be properly reconciled with the various structural dimensions of the material being tested, sharp indentation has become the more preferred method. We therefore focus our present study on sharp indentation, with the full recognition that the tip radius effects of nominally "sharp" indenters need to be carefully accounted for in relation to the depth of penetration of the indenter into the material and the characteristic structural dimensions of the material, so as to avoid the effects of the radius of the sharp indenter tip on the

estimated properties of the materials.

The high sensitivity of traditional brute-force calculations to solve the inverse indentation problem and the uncertainty inherent in such calculations in uniquely extracting elastoplastic properties from indentation responses are known to arise from functional nonlinearity [50, 46, 121]. Furthermore, there are presently no general methods available that can accurately account for tip radius effects on the elastoplastic properties extracted from indentation analyses. This situation is further compounded by the fact that within a portion of the parametric space for the wide spectrum of engineering materials for which instrumented indentation could serve as a useful property assessment tool, the inverse indentation problem may not provide a unique set of predictions for mechanical properties from the indentation data. Therefore, there exists a critical need to explore new ways of determining the elastoplastic properties of materials from depth-sensing instrumented indentation with a greater degree of confidence in their uniqueness, accuracy and fidelity before the potential for the broad adoption of the method for many emerging areas of technology can be fully realized. Furthermore, to establish a scalable method for a wide-variety of applications, and to minimize errors in extracting mechanical properties, it becomes inevitable to assess ways in which the latest developments in deep learning (DL) can be employed to harvest significant improvements for solving the inverse indentation problem.

4.1.2 Recent advances in deep learning and multi-fidelity methods

Recent developments of data-driven methods, such as deep neural networks (NNs), provide us with opportunities that cannot be tackled solely through traditional methods. In addition to well-known applications of machine learning (ML) algorithms in such fields as image/video analysis [106, 200] and natural language processing [228], ML has also been used for various engineering problems, such as in the discovery of new materials [190] and in healthcare [63]. However, data-driven methods usually require a large amount of data to train the neural network model, and in many engineering problems, it is often difficult to obtain necessary data of high accuracy. In these situations, it may be advantageous to complement the dataset of expensive experimental measurements by employing synthetic data derived from simulations of physical models. An example of such an approach entails the use of density functional theory calculations to train neural networks so that DL algorithms can be developed to determine the least energetically expensive means of modulating the bandgap of a semiconductor material through elastic strain engineering [193].

Multi-fidelity methods [109] using Bayesian modeling to integrate high and low resolution simulations can serve as one possible means for training data. This type of data fusion could help to train DL models when limited experimental data that lead only to insufficient levels of accuracy are available. The training data could come from different sources, e.g., from instruments with different resolutions and/or from simulations using different levels of accuracy in predictive capabilities. The multi-fidelity method [109] is hierarchical so that high-fidelity and low-fidelity data can be identified and assigned to train DL algorithms. This

Bayesian multi-fidelity modeling based on Gaussian process regression (GPR) [65] can also alleviate the extreme computational cost of training. However, the GPR method suffers from two critical short-comings: 1) it is computationally prohibitive to manage big datasets, and 2) it is not sufficiently accurate when dealing with nonlinear correlations. To overcome some of these limitations, it is possible to resort to a scalable multi-fidelity approach based on neural networks [146], although the efficacy of such an approach has not yet been tested and validated for applications. In this work, we present a multi-fidelity NN (MFNN) method that is capable of fusing together different sets of data with different fidelity levels, arising from different experimental measurement accuracy or from different levels of sophistication of computational modeling (e.g., two-dimensional [2D] vs. 3D computational simulation and different levels of finite element mesh refinement).

4.1.3 Prior work on ML for computational mechanics and inverse indentation problems

Some prior work has explored use of ML to solve both forward and inverse problems in computational mechanics, and in particular, have trained NNs to extract material properties from instrumented indentation data. The training process in these cases usually involved fitting a numerical simulation dataset. For example, based on data points of spherical indentation load-displacement curves from finite element simulations, a trained NN was established to estimate material parameters [92, 93, 216, 141]. Trained NNs were generated to reproduce the loading portion of sharp nanoindentation load-displacement curves [78]. A NN-based surrogate model was used in order to reduce the number of finite element method

(FEM) conical indentation simulations to extract material properties [127]. Besides NNs, other machine learning approaches have also been employed to solve the indentation problems, such as identification of plastic properties from conical indentation using Bayesian-type analysis [237]. These methods, however, were generally cumbersome to use in practice as they required fitting of the indentation loading (and/or unloading) curves or extensive iterations with finite element simulations. In addition, they were not systematically tested throughout the broad parameter space for a wide variety of engineering materials to establish their predictive capabilities and levels of accuracy. They have also not been extensively validated by comparisons with experimental observations. In summary, the latest advances in deep learning have not yet been fully utilized for solving a highly nonlinear inverse problem, such as that involving an inverse indentation problem.

4.1.4 Objectives of the present study

In the present study, we present a unique combination of the latest developments in deep learning and multi-fidelity methods for datasets with the specific objective of significantly enhancing the accuracy, reliability and predictability of mechanical properties of elastoplastic materials from inverse analyses of depth-sensing instrumented indentation results. Specifically, this study is aimed at achieving the following objectives:

1. Establish new algorithms for solving the inverse problem in instrumented indentation for single, dual and multiple indentation, whereby the elastoplastic properties of the indented material can be predicted with much greater accuracy than currently feasible through traditional brute-force computation

and function-fitting methods that rely on the same set of available training data.

2. Establish a scalable and stable multi-fidelity NN that can

   (a) significantly reduce the required number of high-fidelity data for instrumented indentation to achieve a desired level of accuracy in the prediction of mechanical properties;

   (b) utilize known physical and scaling laws to improve prediction accuracy; and

   (c) integrate simulation data and experimental data for training in order to significantly reduce systematic errors, in indentation analyses, arising from material variability or experimental conditions.

3. Validate the methods presented here through direct comparisons with indentation experiments for several different materials, including two traditionally made (wrought) 6061 and 7075 aluminum alloys and six 3D-printed Ti-6Al-4V alloys.

4.2 Results

We begin here with a graphical illustration of problem statement in Fig. 4.1, showing the forward and reverse analysis of sharp instrumented indentation (Fig. 4.1A), in order to connect our results to the relevant parameters and nomenclature. Representations of the NN architecture are shown for the single-fidelity (Fig. 4.1B) datasets, the multi-fidelity datasets (Fig. 4.1C) without residual connection [146], and the multi-fidelity datasets proposed in this work with residual connection (Fig.

4.1D). In this section, we first show the results using the single-fidelity NN architecture (Fig. 4.1B), to demonstrate how they improve the estimation of mechanical properties of elastoplastic materials compared to those extracted solely from previously established dimensionless fitting functions for inverse analysis of conical indentation [50]. These fitting functions were obtained based on brute-force finite element simulations covering the commonly observed elastic and power-law plastic parameter space for engineering metals [50] and the dimensional analysis of indentation process [42, 50, 43]. This is followed by training the datasets on results from 2D and 3D finite simulations of sharp conical indentation using the finite element method (FEM). Subsequently, we extend the scope of the deep learning analyses to include results from different multi-fidelity approaches. For all these results, we compare them with experimental results involving using the Berkovich indentation on Al-6061, Al-7075 and 3D-printed Ti alloys. More details of the method and nomenclature used in the present study can be found in Methods 4.4 and Appendix C.

4.2.1 Improving inverse analysis results for sharp indentation using single-fidelity NN architecture

Training NNs using data generated from dimensionless fitting functions

To demonstrate that NNs are capable of representing the correlation between (C, $dP/dh, W_p/W_t$) and E^* (or σ_y), where C, dP/dh and W_p/W_t are loading curvature, initial unloading slope, and the ratio of residual plastic work and total work, respectively (see more details in Appendix C), we first generate a dataset using the previously established dimensionless fitting functions. The data points used

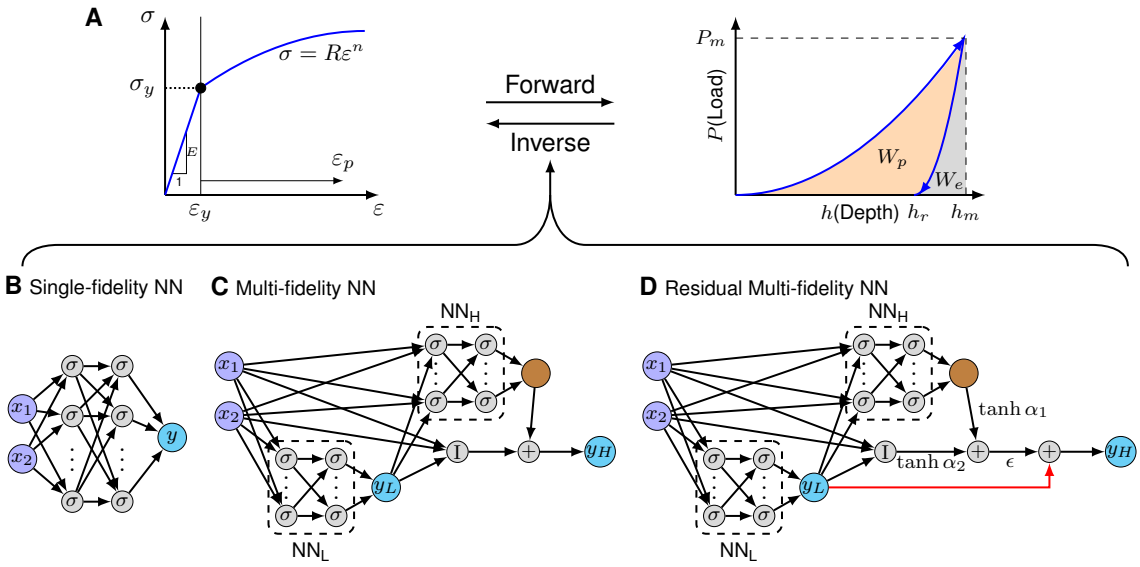


Figure 4.1: Deep learning methods to solve inverse problems in depth-sensing instrumented sharp indentation. (**A**) Schematic illustration of the power-law elastoplastic stress-strain behavior used in the present study (left) and a typical load (P) versus displacement (h) response of an elastoplastic material to instrumented sharp indentation (right). (**B** to **D**) Flowcharts of the neural networks for solving (B) single-fidelity inverse problems, e.g., single indentation, and dual/multiple indentation, and (C and D) multi-fidelity inverse problems involving datasets of different fidelity and accuracy. Input variables such as x_1 and x_2 represent parameters such as C, dP/dh, W_p/W_t, and output variable y represents material properties such as E^* or σ_y. We only show two variables as the neural network inputs for clarity, but the number of inputs could be three or four for single indentation or dual/multiple indentation problems. The NN inputs of all cases and training datasets used are summarized in Appendix C, Tables C.1 and C.2. (C) The original multi-fidelity NN in ref. [146]. (D) The new multi-fidelity NN proposed in this chapter involves a residual connection (red line) from the low-fidelity output y_L to the high-fidelity output y_H. σ and I are the nonlinear and linear activation functions, respectively.

for fitting were obtained through finite element simulations covering commonly observed elastic and power-law plastic parameter space for engineering metals in ref. [50] for conical indentation with a half included-tip-angle of 70.3°. We then train NNs using these data points (Appendix C, Fig. C.1A). The mean absolute percentage error (MAPE) defined as

$$\text{MAPE} = \frac{1}{N}\sum_{i=1}^{N}\left|\frac{A_i - F_i}{A_i}\right|,$$

is calculated against the same dataset, where N is the number of data points, A_i and F_i are the true and prediction values of the i-th data point, respectively. We observe from Fig. C.1A that the errors associated with the extracted values of yield strength σ_y ($\sim 50\%$ or higher) are much larger than those for the effective indentation elastic modulus, E^* (~10%). This is an illustration of the inherently high sensitivity of the inverse problem, especially for plastic properties. As expected, when only data generated from the fitting functions for training are used, the trained NNs do not perform any better than the fitting functions, and only reach the same performance with a high number of training data points.

Training NNs using data obtained from 2D FEM simulations

Next we consider a dataset generated by conical (2D axisymmetric) FEM simulations (see [50] for model setup). The FEM dataset involved simulations for 100 different elastoplastic parameter combinations, and we removed 3 data points with $n > 0.3$ and $\sigma_y/E^* \geq 0.03$, where the inverse problem may have non-unique solutions. The possible non-uniqueness comes from the fact that increasing elastic modulus, plastic yield strength or strain-hardening exponent can all result in an increased loading curvature, with the consequence there may exist multiple

elastoplastic parameter sets in achieving nearly identical indentation loading/unloading curves (see more detailed discussions in ref. [50]). The green curves with solid square symbols in Fig. 4.2A and B (also see Fig. C.1B) show the results of training NNs for E^* and σ_y using different numbers of conical indentation FEM simulation data points. By using merely 20 training points for E^*, the trained NN already performs better than the previously established fitting functions in ref. [50]. For σ_y, 50 or more data points are required to achieve better accuracy than the previous algorithm established by direct fitting of the finite element data points [50]. With 80 data points for training, the average error for E^* can be improved to ~5% significantly lower than ~8% from using the algorithm established in ref. [50].

Training NNs using FEM data obtained from multiple indenter geometries

Fig. 4.2 shows the results of training NNs for E^* and σ_y using 2 or 4 indenters with different tip geometries. The trained 2-indenter and 4-indenter NNs perform better than the single-indention NNs shown in Fig. C.1B. More indenter geometries improve accuracy. With a large enough size of training datasets (≥ 20 for E^*, ≥ 90 for σ_y with 2 indenters; and ≥ 60 for σ_y with 4 indenters), the trained NNs begin to outperform the dual indentation algorithm [46]. For the trained 2-indenter NNs, the average error for E^* is about 2% — much better than that achieved in ref. [50] or ref. [46] using traditional fitting functions. For the trained 4-indenter NNs, the average error for E^* is $< 2\%$, and for σ_y, the average error becomes $< 7\%$. Note that with traditional algorithms, it is not straightforward to utilize "redundant" information for solving unknown variables, while in the case of training the 2-indenter or 4-indenter NNs more/redundant input variables can be easily utilized to improve accuracy in the estimation of elastic and plastic properties.

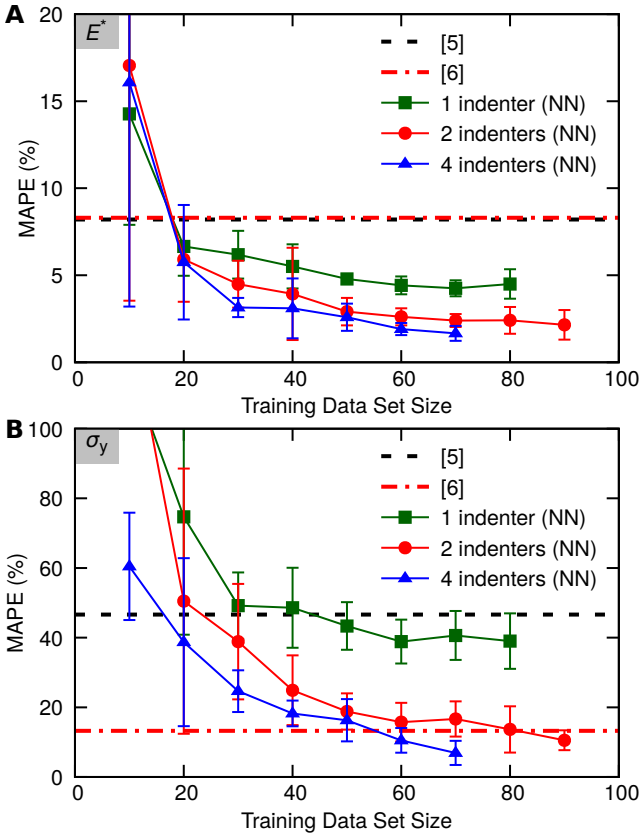


Figure 4.2: Results of mean absolute percentage error (MAPE) as a function of the Training Dataset Size for dual and multiple indenters. Results of training NNs for (**A**) E^* and (**B**) σ_y using computational simulations of sharp conical indentation using FEM with 1, 2 or 4 different indenter tip geometries. The black dotted line and the red dash-dotted line show the average error of directly applying the single-indentation fitting functions in ref. [50] and dual-indentation fitting functions in ref. [46], respectively. The 1-indenter data are obtained using conical tip with 70.3° half-included tip angle; the 2-indenters data are obtained using conical tips with 70.3° and 60° half-included tip angles; and the 4-indenters data are obtained using conical tips with 70.3°, 50°, 60° and 80° half-included tip angles.

4.2.2 Inverse analysis using multi-fidelity NNs

Approach 1: Integrating data generated from fitting functions (low fidelity) and FEM simulation data (high fidelity)

In this example, we test the multi-fidelity approach for the conical single indentation data using only materials with $n \leq 0.3$ (which still covers the material parameter space for a wide spectrum of engineering metals and alloys), as shown in Figs. 4.3A and B. The low-fidelity data use 10,000 (for E^*) or 100,000 (for σ_y) data points from the formulas in ref. [50], and the high-fidelity data are from the finite element simulations. All data in Figs. 4.3A and B pertain to a conical (2D axisymmetric) indenter with a half-included tip angle of 70.3°. By using the multi-fidelity approach, (i) higher accuracy is achieved compared to using only high-fidelity data, and (ii) the number of high-fidelity data points required for achieving high accuracy is also significantly reduced. Only 10 high-fidelity data points are needed to achieve 5% average error for E^*, and only 40 high-fidelity data needed to achieve better accuracy for σ_y than the traditional fitting functions [50].

Here we compare our multi-fidelity neural network (MFNN) method using the residual connection technique introduced in this chapter (Fig. 4.1C, see more details in Methods) with the MFNN used in ref. [146] without the residual connection (Fig. 4.1B). We test the performance of these two methods on the property predictions based on multi-fidelity datasets as described in Section 4.4. Note that low fidelity refers to data obtained using the fitting functions and high fidelity refers to 2D FEM data. Appendix Fig. C.2A (blue line) shows that the training of the original MFNN is quite unstable with a large standard derivation. When comparing the mean error of E^* in Appendix Fig. C.2A (blue and red lines), note

that the error becomes more stable, i.e., the training of NNs becomes more stable with the addition of residual connection. For those well-trained networks, the error in E^* is similar. Furthermore, as shown in Appendix Fig. C.2B, our proposed MFNN with the residual connection also achieves higher accuracy than the fitting functions for the case of σ_y, in addition to being more stable.

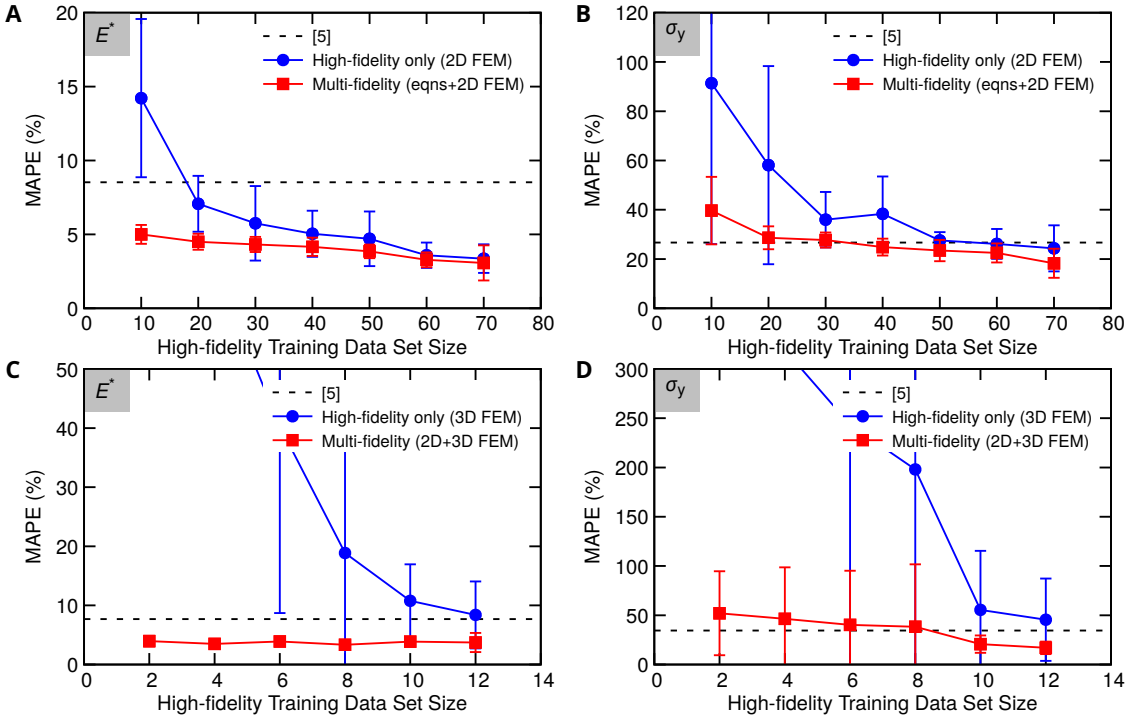


Figure 4.3: Mean average percentage error as a function of training dataset size for multi-fidelity NNs trained by 2D and 3D FEM simulations of inverse indentation. (**A** and **B**) Results of multi-fidelity NNs trained by integrating low-cost low-fidelity data using fitting functions [50] together with limited number of high-fidelity FEM data for (A) E^* and (B) σ_y. In (A) and (B), the low-fidelity data use 10,000 (for E^*) or 100,000 (for σ_y) data points from the formulas in ref. [50]. All 2D axisymmetric FEM data are assuming a conical indenter with a half-included tip angle of 70.3°. (**C** and **D**) Results of multi-fidelity NNs trained by integrating 2D axisymmetric FEM results (low fidelity) together with 3D FEM simulation data (high fidelity) for (C) E^* and (D) σ_y. The low-fidelity 2D FEM data in (C) and (D) include 97 axisymmetric FEM simulations with different elastoplastic parameters. All 3D FEM data are using a 3D Berkovich indenter, which has a three-sided pyramid sharp tip that can maintain its self-similar geometry to very small indentation depth. The Berkovich indenter has a half angle of 65.3°, measured from the tip axis to one of the pyramid surfaces.

Approach 2: Integrating 2D axisymmetric FEM data (low fidelity) and 3D FEM data (high fidelity)

Figs. 4.3C and D show the results of MFNNs trained by integrating 2D axisymmetric FEM simulation results (low fidelity) together with 3D FEM simulation data for a Berkovich indenter (high fidelity) to estimate E^* and σ_y. Here MAPE is calculated with respect to the 3D FEM data, which have higher fidelity than the 2D axisymmetric FEM data. Although conical indentation finite element results with a 70.3° included half-angle are considered good approximations of the actual indentation results from a 3D Berkovich or Vickers indenter tip [50, 46, 121, 28, 31, 220], significant errors can still occur due to the inherent high sensitivity of the inverse problem, especially for extracting plastic properties, as shown in Figs. 4.3C and D. The foregoing results illustrate that the multi-fidelity approach leads to much more accurate estimates of mechanical properties from instrumented sharp indentation data with a smaller number of high-fidelity data points than both the single-fidelity approach and the fitting functions.

Approach 3: High-fidelity training datasets for error correction

Here we first test the trained NNs obtained above (Approach 2) for the Berkovich indenter tip for two indentation experimental datasets from traditional (wrought) Al alloys Al6061-T6511 (6 experiments) and Al7075-T651 (6 experiments) with the indentation characteristics summarized in Appendix, Table C.3. The indentation raw datasets used are the same as those used in ref. [50], and $\frac{dP}{dh}|_{h_m}$ is evaluated by the best linear fitting within 5% of each unloading curve. The elastoplastic properties of Al6061-T6511 are Young's modulus E = 66.8 GPa (E^* = 70.2 GPa),

yield strength σ_y = 284 MPa and strain-hardening exponent n = 0.08; and the properties of Al7075-T651 are E = 70.1 GPa (E^* = 73.4 GPa), σ_y = 500 MPa and n = 0.122. In addition, to reduce the incurred systematic experimental errors we use NNs to learn from three randomly selected experimental data points added as high-fidelity data in the NN training process in multi-fidelity Approach 2. Specifically, the low-cost 2D axisymmetric finite element datasets are still used as low-fidelity data, and the limited number of 3D Berkovich indentation finite element data are used together with 3 additional experimental data points as high-fidelity data for the case of Al6061-T6511 or Al7075-T651 alloy. There are up to 20 unique combinations for randomly selecting 3 out of 6 experiments in each case. Here the results are obtained by exhausting all 20 possibilities.

Fig. 4.4A summarizes the inverse analysis results using different approaches. The NNs trained by 2D axisymmetric FEM results (low fidelity) together with 3D FEM simulation data (high fidelity) perform better than the previous established equations in ref. [50]. The NNs trained by adding experimental results as high-fidelity training data to the 2D and 3D FEM data perform very well for both E^* and σ_y with MAPE less than 4% for both Al6061-T6511 and Al7075-T651, leading to significantly improved accuracy for σ_y with this "hybrid" multi-fidelity approach. Assuming power-law strain-hardening behavior, our proposed method can also be used to extract strain-hardening characteristics from instrumented indentation. To achieve that, we first train NNs to predict stresses at different plastic strains, and then compute the strain-hardening exponent by least-squares fitting of the power-law hardening function.

Fig. 4.4B shows the inverse analysis results of using multi-fidelity NNs to extract additional data points from the stress-strain curve (i.e. to determine strain-hardening behavior), where selected stress values at 3.3%, 6.6% and 10% plastic

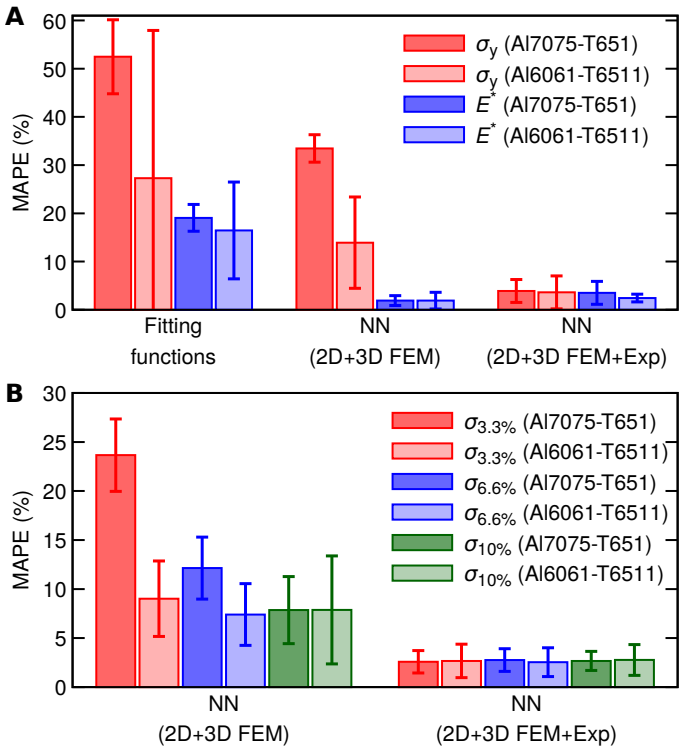


Figure 4.4: Inverse analysis results of mean average percentage error (MAPE) of (**A**) E^* and σ_y, and (**B**) $\sigma_{3.3\%}$, $\sigma_{6.6\%}$ and $\sigma_{10\%}$ for two aluminum alloys Al6061-T6511 and Al7075-T651 (here the subscripts 3.3%, 6.6% and 10% for σ represent plastic strains). The results labeled as "fitting functions" are obtained directly using previously established equations in ref. [50]. The results labeled as "NN (2D+3D FEM)" are obtained using NNs trained by integrating 2D axisymmetric FEM data (low fidelity) with 3D Berkovich FEM data (high fidelity), and the results labeled "NN (2D+3D FEM+EXP)" are obtained using NNs trained by adding experimental results as high-fidelity training data in addition to the 2D and 3D FEM training data.

strains are obtained. The NNs trained by adding experimental results as part of the high-fidelity training data also perform very well for $\sigma_{3.3\%}$, $\sigma_{6.6\%}$ and $\sigma_{10\%}$ with MAPE less than 4% for both Al6061-T6511 and Al7075-T651, significantly improving the accuracy for evaluating stresses at different plastic strain using the "hybrid" multi-fidelity approach. Fig. 4.5 shows the corresponding stress-strain curves obtained by least-square fitting of the power-law hardening behavior, exhibiting good matching of the experimental data (with experimentally extracted hardening exponent n = 0.08 and 0.122), whereas n = 0.073 and 0.127 for Al6061-T6511 and Al7075-T651, respectively, estimated using the "hybrid" multi-fidelity approach. Note that when hardening is low (i.e. $n \rightarrow 0$), directly estimated errors of n can be misleading because very small variations in hardening response can lead to large fractional errors for elastic-perfectly plastic metal alloys. Comparing errors in stresses at different plastic stains is a more objective way in evaluating the accuracy with respect to the stress-strain behavior or the hardening behavior.

Next we test the NN algorithms on an extensive set of experiments performed on six different, 3D-printed, Ti-6Al-4V alloys. Full details of the 3D-printing conditions, the ensuing microstructures of the six Ti alloys, and the tensile stress-strain characteristics are all available in ref. [116]. The six 3D-printed Ti alloys with different microstructures are designated as B3067, B3090, B6067, B6090, S3067 and S6067 [116]. We carried out depth-sensing instrumented nanoindentation experiments of these 3D-printed Ti alloys using the method described in Methods. There are 144 repeated indentations conducted for each 3D-printed Ti alloy. We perform inverse analyses of these six alloys and estimate their elastoplastic properties using the various machine learning approaches introduced in this chapter. We then compare these predictions with direct and independent experimental assessments of the elastoplastic properties of the six alloys from the tensile stress-strain responses

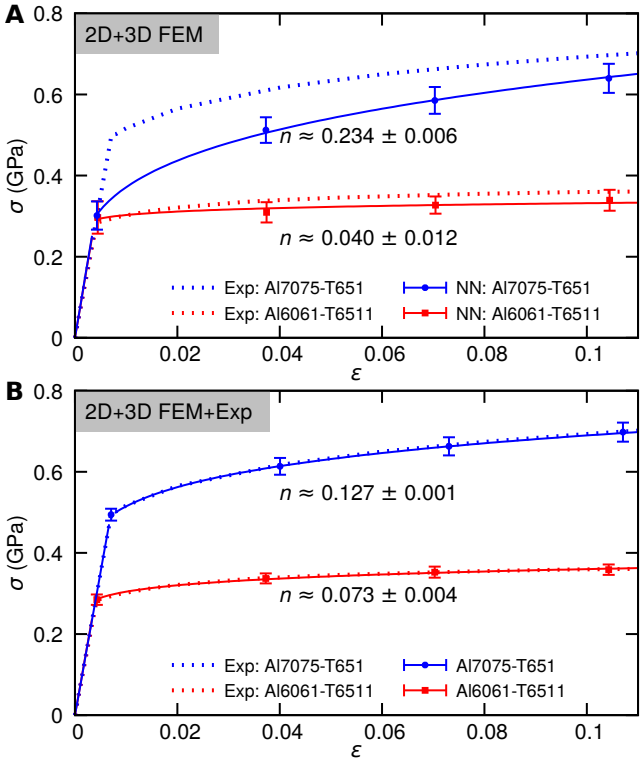


Figure 4.5: Inverse analysis results of hardening exponent for two aluminum alloys Al6061-T6511 and Al7075-T651. The hardening exponent is obtained by least squares fitting of σ_y, $\sigma_{3.3\%}$, $\sigma_{6.6\%}$ and $\sigma_{10\%}$ predicted by NNs trained by (**A**) 2D and 3D FEM data, and (**B**) 2D, 3D FEM data and 3 experimental data points. Here the subscripts 3.3%, 6.6% and 10% for σ represent plastic strains.

obtained in ref. [116]. Appendix Table C.4A summarizes the indentation characteristics of six 3D-printed Ti-6Al-4V alloys extracted directly from raw indentation curves. Appendix Table C.4B lists the indentation characteristics of two 3D-printed titanium alloys from indentation curves corrected with an estimated indenter tip radius of 0.6 μm (see details of the tip radius estimation and tip-radius-effect correction method in Appendix C, based on the method suggested in ref. [45]), all using the same experimental setup with a maximum indentation load of 9 mN for each indentation experiment. The yield strength values, σ_y, of B3067, B3090, B6067, B6090, S3067 and S6067 are 1121, 1168, 1102, 1151, 1121 and 1063 MPa, respectively, and the nominal Young's modulus of these 3D-printed Ti alloys is E = 110 GPa (E^* = 109.6 GPa).

For indentations made on Ti-6Al-4V (B3067), Fig. 4.6 summarizes the inverse analysis results using the different approaches introduced in this chapter for both E^* and σ_y. The results labeled as "NN (raw)" and "NN (tip)" are obtained by applying NNs trained by integrating 2D axisymmetric FEM data (low fidelity) with 3D Berkovich FEM data (high fidelity), either by directly applying the raw indentation data or by using the indentation data after correcting the raw data for the indenter tip radius effects, respectively. NNs trained using only FEM data, when operating directly on raw indentation data, exhibit medium accuracy of 24.2 ± 4.6% MAPE in estimating the elastic modulus, but an unacceptably high MAPE of 105.5±16.7% in evaluating σ_y. However, when tip-radius-effect corrected indentation data are used in the analyses, we observe significantly reduced inverse analyses errors for both E^* (MAPE = 5.4 ± 3.1%) and σ_y (MAPE = 40.3 ± 8.3%). With tip-radius-effect correction, we find that the extracted values of E^* and σ_y are much closer to the uniaxial test results shown in Fig. C.1 in ref. [116]. When using the "NN (raw)" results, it is evident that systematic bias occurs in the extraction

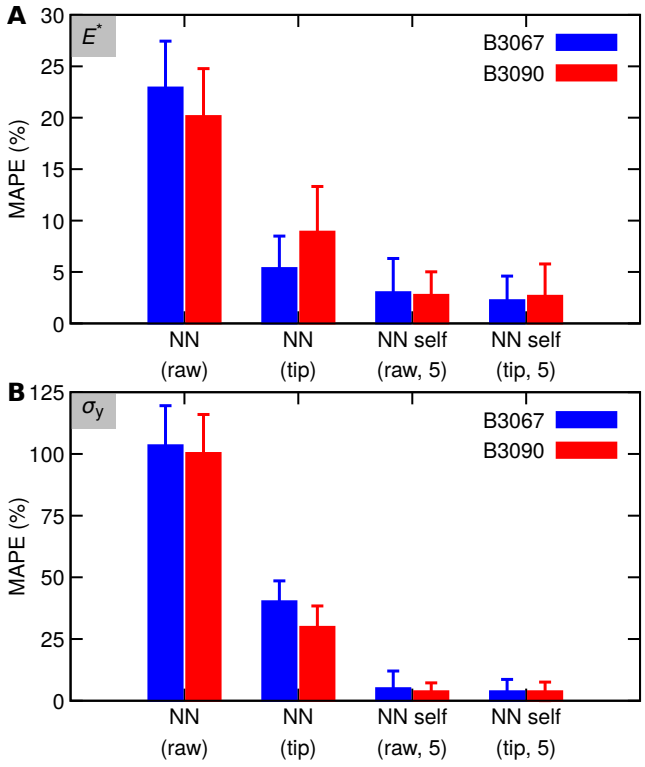


Figure 4.6: Inverse analysis results of (**A**) E^* and (**B**) σ_y for two 3D-printed Ti-6Al-4V alloys B3067 and B3090. The results labeled as "NN (raw)" and "NN (tip)" are obtained by applying NNs trained by integrating 2D axisymmetric FEM data (low fidelity) with 3D Berkovich FEM data (high fidelity), using directly the raw indentation $P - h$ data and using the tip-radius-effect corrected indentation data, respectively. The results labeled "NN self (raw, 5)" and "NN self (tip, 5)" are obtained by applying NNs trained by adding 5 randomly picked experimental indentation curves as high-fidelity training data in addition to the 2D and 3D FEM training data, using directly the raw indentation data and using the tip-radius-effect corrected indentation data, respectively. Full details of the experimental data on instrumented indentation and stress-strain response for both B3067 and B3090, along with the conditions for 3D printing and depth-sensing indentation, and microstructure evolution can be found in ref. [116]; a brief summary of these literature data is provided in the Appendix C.

of both E^* and σ_y; from the "NN (tip)" results, there appears to be systematic bias for σ_y, even after the significant improvement in predictive capability by applying tip-radius-effect correction.

We now apply our multi-fidelity Approach 3 described earlier in an attempt to further reduce systematic errors in the inverse analyses. For this purpose, we randomly select five experimental data points out of 144 as additional input to high-fidelity data in the NN training process. Specifically, the low-cost 2D axisymmetric finite element datasets are still used as low-fidelity data, while the limited number of 3D Berkovich indentation finite element data are used together with 3 additional experimental data points as high-fidelity data. The results are pooled together by exploring the full spectrum involving 10 uniquely different ways of random selection of such data. In Fig. 4.6, the results labeled "NN self (raw, 5)" and "NN self (tip, 5)" are obtained by applying NNs (trained by adding the 5 randomly selected B3067 experimental indentation curves as high-fidelity training data in addition to the 2D and 3D FEM training data) to the raw indentation data and to the tip-radius-effect corrected indentation data, respectively. NNs, trained using the "hybrid" multi-fidelity approach that included the added experimental training data now significantly reduce errors when operating directly on raw indentation data and when operating on tip-radius-effect corrected indentation data. Specifically, for E^* MAPE drops to 3.0 ± 3.3% and 2.3 ± 2.4% for raw indentation data and tip-radius corrected data, respectively, and for σ_y MAPE drops to 5.1 ± 7.0% and 3.9 ± 4.8% respectively. Although NNs operating on tip-radius-effect corrected data still perform better, the "hybrid" multi-fidelity approach introduced here is found to be substantially more effective in learning from the data and in correcting errors from tip radius effects and other systematic biases arising from uncorrected raw data. Similar results are shown for indenta-

tions made on another 3D-printed Ti-6Al-4V alloy (B3090) in Fig. 4.6 for both E^* and σ_y.

Finally, we test a more practically useful variation of "hybrid" multi-fidelity approach. Here we aim to reduce systematic errors by randomly selecting indentation experimental data points from a different calibration material (while using the same experimental/post-processing setup) as added high-fidelity data in the NN training process in multi-fidelity Approach 3. Specifically, the low-cost 2D axisymmetric finite element datasets are still used as low-fidelity data while the limited number of 3D Berkovich indentation finite element data are used together with some additional experimental data points from a different calibration material B3067 (with 1 to 20 data points randomly selected from a total of 144 data points) as high-fidelity training data; the trained NNs are used to analyze B3090 indentation data.

Fig. 4.7 summarizes the indentation inverse analysis results for B3090 using this approach (denoted as "Peer") as compared to the results where the added high-fidelity training data are from the same material (B3090) with the same experimental conditions (denoted as "Self"). Here MAPE (log scale) is plotted against the number of randomly selected experimental training data, n_{exp} (linear scale) from either the same material ("Self") or from another Ti alloy ("Peer"). Except when no experimental data are added for training (at 0), each data point in Fig. 4.7 represents the results pooled together from 10 uniquely different ways of random selection. The notion of "(raw)" and "(tip)" in the labels indicate all experimental data used are either the uncorrected raw indentation data or those corrected for tip radius effects, respectively. It is clear from Fig. 4.7 that adding experimental training data from the same material ("Self") or from a different calibration material ("Peer") under the same experimental conditions

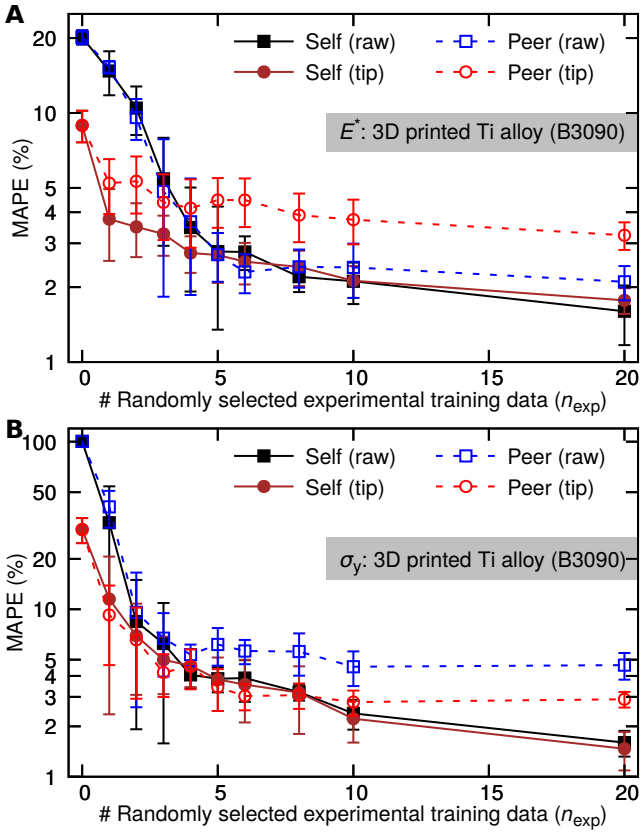


Figure 4.7: Inverse analysis of a 3D printed Ti-6Al-4V alloy B3090. (**A**) E^* and (**B**) σ_y versus the number of randomly selected experimental training data either from B3090 (denoted as "Self") or from B3067 indentation experiments ("Peer"). The notion of "(raw)" and "(tip)" in the labels indicate all experimental data used are from the uncorrected raw indentation data and the tip-radius-effect corrected indentation data, respectively.

can significantly reduce systematic errors for both E^* (MAPE < 4%) and σ_y (MAPE < 5%). However, we note that the benefits of adding experimental training data begins to saturate when $n_{exp} \geq 5$. With sufficient training data, the "hybrid" multi-fidelity approach shows remarkable ability to learn and correct from the raw data any systematic errors from tip radius effects and other factors directly. As expected, adding experimental training data from the same material ("Self" cases) normally performs better than adding from a different calibration material ("Peer" cases) under the same experimental conditions; in particular, we observe error reduction at $n_{exp} = 20$ by as much as one and two orders of magnitude for estimating E^* and σ_y, respectively, from the inverse analysis.

Fig. 4.8 and Appendix C Fig. C.4 summarize the inverse analysis results of MAPE for E^* and σ_y for 3D-printed Ti-6Al-4V alloys (B3067, B3090, B6067, B6090, S3067 and S6067) as a function of n_{exp} for randomly selected experimental training data from B3067 indentation experiments. Except when no experimental data are added for training (at 0), each data point represents the results pooled together from 10 uniquely different ways of such random selection. All indentation experimental data used are from the uncorrected raw indentation data. The black dashed line is the "Self" training case for B3067, while the curves representing other colors are the "Peer" training cases using B3067 indentation data for training. Again, all the trends noted in Fig. 4.7 are also observed here in Fig. 4.8 and Appendix C Fig. C.4, showing the general applicability of this "hybrid" multi-fidelity approach by adding "Peer" experimental data as high-fidelity training data for improved inverse analysis accuracy.

Assuming, once again, power-law strain-hardening behavior, we can evaluate stresses at different plastic strain values, and then compute the strain-hardening exponent by least-squares fitting of the power-law hardening function for 3D-

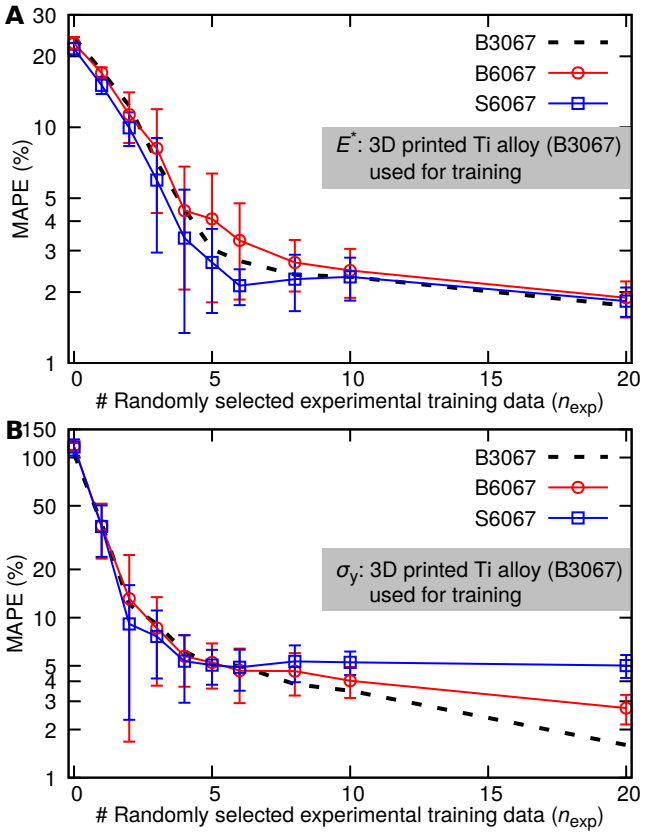


Figure 4.8: Inverse analysis of three 3D-printed Ti-6Al-4V alloys. (**A**) E^* and (**B**) σ_y for 3D-printed Ti-6Al-4V alloys (B3067, B6067, and S6067) versus the number of randomly selected experimental training data from B3067 indentation experiments. All experimental data used are from the uncorrected raw indentation data. See also Appendix C Fig. C.4 for additional comparisons.

printed titanium alloys. Fig. 4.9 shows the inverse analysis results of selected stresses at 0% (i.e. σ_y), 0.8%, 1.5% and 3.3% plastic strains and the fitted stress-strain curves for two 3D-printed Ti-6Al-4V alloys using the "hybrid" multi-fidelity approach. Analogous to evaluating the yield strength (stress at zero plastic strain), our predicted stress-strain curves are close to the experimental curves when a few experimental data points are added as part of the high-fidelity data for the training of NNs.

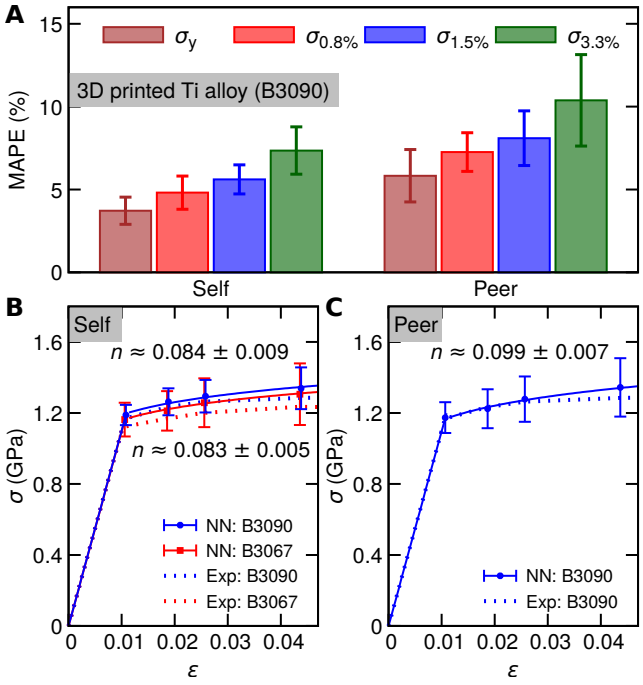


Figure 4.9: Inverse analysis results of hardening exponent for two 3D-printed Ti-6Al-4V alloys B3067 and B3090. (**A**) Mean average percentage error of σ_y, $\sigma_{0.8\%}$, $\sigma_{1.5\%}$ and $\sigma_{3.3\%}$ for B3090 predicted by NNs trained by 2D axisymmetric FEM data (low fidelity) with 3D Berkovich FEM data and 5 randomly picked "self" and "peer" experimental indentation curves (high fidelity). (**B** and **C**) The hardening exponent is obtained by least squares fitting of σ_y, $\sigma_{0.8\%}$, $\sigma_{1.5\%}$ and $\sigma_{3.3\%}$ for (B) "self" and (C) "peer" experimental indentation curves. The experimentally extracted best fit hardening exponent is $n = 0.068$ for both B3090 and B3067 uniaxial experiments, i.e. near zero low hardening. With additional experimental data added for training, the NNs predicts accurately the yield strength and low hardening behaviors. Here the subscripts 0.8%, 1.5% and 3.3% for σ represent plastic strains.

4.2.3 Transfer learning

In the results presented so far, the "hybrid" training of neural networks for each aluminum alloy and each 3D-printed titanium alloy is conducted with a fresh start without any direct connections to the other trained neural networks. On the other hand, to speed up the training of neural networks, we have also developed a transfer learning technique, where the whole multi-fidelity network (both low- and high-fidelity sub-networks) is first trained using all the 2D and 3D FEM data as baseline training. Next given the additional new experimental data, only the high-fidelity sub-network is further trained using these additional experimental data points. The errors from the networks before and after transfer learning are shown in Fig. 4.10. This figure indicates that we can first establish a comprehensive baseline training and then add additional case-specific training later for improved training efficiency and accumulated learning.

4.3 Discussion and concluding remarks

We have demonstrated in this work a general framework for extracting elastic and plastic properties of engineering alloys through a suite of unique approaches that combine the latest advances in depth-sensing instrumented indentation with computational simulations of the mechanical properties of materials and the latest developments in deep learning using neural networks. We have shown how different single-fidelity and multi-fidelity approaches can be customized to extract different levels of accuracy, even when only a small set of training data is available. Furthermore, our method establishes how long-recognized and hitherto unaddressed limitations of extracting plastic properties of materials from indentation

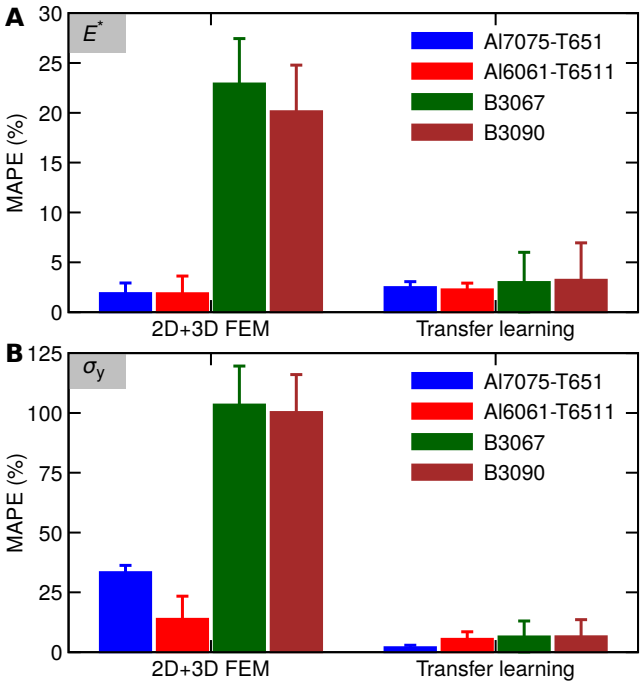


Figure 4.10: Inverse analysis results of (**A**) E^* and (**B**) σ_y for neural networks via transfer learning for two aluminum alloys Al6061-T6511 and Al7075-T651 and two 3D-printed Ti-6Al-4V alloys B3067 and B3090. A multi-fidelity neural network is first trained on the dataset of the 2D FEM as the low-fidelity data and 3D FEM as the high-fidelity data (the results labeled as "2D+3D FEM"). Next the high-fidelity sub-network is continued to be trained using 3 and 5 experiment data for aluminum alloys and 3D-printed Ti-6Al-4V alloys, respectively, which the low-fidelity sub-network does not change (the results labeled "Transfer learning").

data, such as uniqueness of the estimated property values, systematic errors and uncertainties arising from the effects of tip radius of nominally "sharp" indenters, can be overcome to produce a significantly higher level of accuracy and fidelity in the inverse analysis approach.

We have introduced in this work three multi-fidelity approaches, along with single, dual and multi-indenter analyses, with the goal of significantly reducing the required number of high-fidelity datasets to achieve a chosen level of accuracy, and to significantly improve the accuracy and reliability of the mechanical properties extracted from depth-sensing instrumented indentation. Specifically, the methods outlined here, are shown to:

1. significantly reduce the number of high-fidelity datasets needed to achieve a chosen level of accuracy;
2. utilize previously established physical and scaling laws to improve the accuracy and training efficiency; and
3. integrate simulation data and experimental data (i.e., data fusion) for training and significantly reducing material and/or experimental setup related systematic errors.

Our results are validated with independent experimental data and sources for different types of wrought aluminum and 3D-printed titanium alloys.

In order to expedite the training of NNs, we have also developed a transfer learning technique, where the entire multi-fidelity network (both low- and high-fidelity sub-networks) is first trained using all the 2D and 3D FEM data as the baseline training. This baseline training covers the material parameter space for

the majority of engineering metals under an idealized testing condition. When given the additional experimental data for materials under a specific experimental setup, only the high-fidelity sub-network needs to be further trained. The results in Fig. 4.10 show that we can first establish a comprehensive baseline training and then add additional case-specific training later for improved training efficiency and accumulated learning. With a small number of high-fidelity experimental data points added for training, significant improvements are achieved. This is due to that fact that, for a nominally homogeneous material, instrumented indentation experiments are known to produce highly repeatable force vs penetration depth curves when using the same indenter instrument and the same setup. When we have a reliable method that can effectively learn and correct systematic errors, we can then use the method to calibrate the indenter and obtain reliable inverse analyses results. Additional discussion on the sensitivity of the inverse indentation problem in extracting plastic properties can be found in Appendix C and Fig. C.5.

There are several potentially appealing consequences of the results obtained in this work.

1. The approach described here provides unique pathways to extract critically needed information on mechanical properties, which cannot be easily obtained by other means, in a wide variety of engineering applications involving both structural and functional materials of different types and size scales.

2. With the cost-effectiveness and sophistication of instrumented indentation, robotics and computing tools, the present approaches can readily be incorporated in a wide variety of manufacturing settings (such as those involving 3D printing) for in situ and real-time estimation of material properties.

3. The approach is also highly adaptable and dynamic in that refinements in the choice of a particular deep learning approach can be made "on-the-fly" depending on the processing conditions, specimen geometry, material characteristics, speed of manufacturing, and the level of accuracy sought in the extracted values of properties.

4. The approaches described here can also be further enhanced, with appropriate modifications, to account for such factors as (a) the build-up of residual stresses during the processing of the material, (b) level of anisotropy in material properties, (c) multi-component architectures involving particle-reinforced, fiber-reinforced or layered composite materials, and (d) tailoring of surface and bulk properties through the deliberate introduction of structural, compositional, geometrical and property gradients.

4.4 Methods

4.4.1 General considerations

We implicitly utilize physically-based scaling laws such as Kick's law [103] to simplify the problem and reduce data noise. For this purpose, instead of the common practice of directly using data points within the individual indentation curves for training, we choose key indentation parameters such as loading curvature C, initial unloading slope, $\frac{dP}{dh}|_{h_m}$, plastic work ratio, W_p/W_t, etc., for indentation inverse problem input and the power-law elastoplastic material parameters Young's modulus, E (or reduced modulus, E^*), yield strength, σ_y, hardening exponent, n, etc., (defined in eqns (C.1)-(C.4) in Appendix C) as the output.

On the other hand, since different datasets are obtained for different maximum depth of indenter penetration into the substrate, h_m, we apply the scaling law (established through dimensional analysis in ref. [50]) between $\frac{dP}{dh}|_{h_m}$ and h_m:

$$\frac{dP}{dh}|_{h_m} \propto h_m$$

to scale all the datasets with different h_m, thereby reducing the required amount of training data.

4.4.2 NN architecture for single indentation and dual/multiple indentation inverse problems

Single indenter inverse problem

For solving the single indentation inverse problem, two fully-connected NNs are trained separately to represent the mapping from $(C, dP/dh, W_p/W_t)$ to E^* and σ_y, respectively. Each NN has three layers with 32 neurons per layer (Fig. 4.1B). The nonlinear activation function is chosen as the scaled exponential linear unit (SELU) [113]. To avoid over-fitting, regularization can be applied to limit the freedom of the model by adding a penalty on the involved model parameters. Here we use the standard L_2 regularization [165] with a strength of 0.01. Throughout our work, the level of accuracy in estimating any mechanical property is quantified by the mean absolute percentage error (MAPE) [51]. The NNs are optimized using the Adam optimizer [112] with learning rate 0.0001 for 30000 steps.

Dual/multiple indenter inverse problem

For solving the dual/multiple indentation inverse problem, there exist different possibilities for selecting the input parameters. In the present study, we choose $(C, dP/dh, W_p/W_t)$ extracted from indentation curves using the 70.3° conical tip and C from indentation curves using the 60° conical tip as the inputs of NNs. For solving the multiple indentation inverse problem, we choose $(C, dP/dh, W_p/W_t)$ extracted from indentation curves using the 70.3° conical tip and loading curvatures (C) from indentation curves using 50°, 60° and 80° conical tips as the inputs of NNs. For the dual/multiple indentation problem, the same NN architecture is applied as that is utilized in solving the single indentation inverse problem (Fig. 4.1B), except that a faster learning rate of 0.001 is taken.

4.4.3 Experimental method for obtaining indentation datasets from 3D-printed Ti alloys

Nanoindentation experiments were performed on samples (5 mm by 5 mm square cross-section and 10 mm height) that were electro-discharge-machined from larger 3D-printed coupons with selected printing (laser melting) conditions (see ref. [116] for details on the printing conditions and post-printing heat treatment for B3067, B3090, B6067, B6090, S3067 and S6067 samples). The cut samples were first polished using emery paper (particle size = 9 μm) and then electro-polished before indented using a Hysitron Triboindeter (Hysitron, Minneapolis, USA) equipped with a Berkovich diamond indenter tip in load control mode, under the following experimental conditions: peak load = 9 mN, loading rate = 0.9 mN/s, hold time at the peak load = 5 s, unloading rate = 1.8 mN/s. A total of 144 nanoindentations

were performed on each sample over a 360 × 360 μm^2 area, with a distance of 30 μm (in both x and y directions) between two adjacent indents. Before each set of nanoindentation experiments, the tip was calibrated using a standard reference sample of fused quartz. Load, P, versus depth of penetration, h, data were recorded.

4.4.4 Multi-fidelity NN and unique inverse problem setups

Residual-based Multi-fidelity NN

For the multi-fidelity neural network (MFNN), we propose a new residual-based MFNN (Fig. 4.1D), extending the method first developed by Meng and Karniadakis [146] (Fig. 4.1C). As shown in Figs. 4.1C and D, the low fidelity function y_L is the output of a neural network $\mathrm{NN_L}$ with input x. In ref. [146], the high fidelity y_H is a weighted summation of a linear function and a nonlinear function (Fig. 4.1C):

$$y_H(x) = \alpha_1 f_{linear}(x, y_L(x)) + \alpha_2 f_{nonlinear}(x, y_L(x)),$$

where $f_{linear}(x, y_L)$ and $f_{nonlinear}(x, y_L)$ are linear and nonlinear functions of inputs (x, y_L), respectively. In particular, $f_{nonlinear}(x, y_L)$ is another NN, represented by $\mathrm{NN_H}$ in Fig. 4.1C, while $f_{linear}(x, y_L)$ is a single neuron with no activation function. In addition to the parameters in $f_{linear}(x, y_L)$ and $f_{nonlinear}(x, y_L)$, α_1 and α_1 are also two additional parameters to be trained.

We extended this method by adding an extra connection from y_L to y_H, and adopting a specific form of α_1 and α_2 to correlate the high- and low-fidelity data

(Fig. 4.1D):

$$y_H = \alpha_L y_L + \epsilon(\tanh \alpha_1 \cdot f_{linear}(x, y_L) + \tanh \alpha_2 \cdot f_{nonlinear}(x, y_L)),$$

where α_1 and α_2 are two parameters to be learned. The coefficient α_L represents the ratio of the high-fidelity to low-fidelity outputs, and ϵ represents the ratio of the residual to the high-fidelity output. In principle, α_L can also be a learnable parameter as α_1 and α_2, but here we choose α_L to be 1, because in the indentation problems we considered y_L is usually of the same order of y_H, i.e., the residual $y_H - y_L$ is much smaller than y_L and y_H. For the same reason, we choose ϵ as a small positive number to be 0.1. The network prediction is not very sensitive to the values of α_L and ϵ, if their magnitudes are chosen correctly. However, the network may be trained to a wrong state if the values are incorrectly selected. In addition, at the beginning of training, we initialize α_1 and α_2 to be 0, such that the learning of y_H starts from y_L.

Our proposed MFNN makes the training process more stable and yields better accuracy compared to the original MFNN. The reason is that it is much easier for $f_{linear}(x, y_L)$ and $f_{nonlinear}(x, y_L)$ to learn the residual $y_H - y_L$ than to learn y_H directly. This residual approach was first proposed with the name "ResNet" [87], and since then it has been widely used in many computer vision tasks.

We follow three multi-fidelity machine learning approaches in the present study. All versions of multi-fidelity neural networks are implemented in DeepXDE [136], a user-friendly Python library designed for scientific machine learning.

Approach 1: Integrating data generated from fitting functions (low fidelity) and FEM simulation data (high fidelity)

We first test the proposed multi-fidelity approach using the conical single indentation data for materials with $n \leq 0.3$ (covering the material parameter space for majority of engineering metals). The low-fidelity dataset is generated by using the fitting functions from ref. [50] while the high-fidelity dataset is based on the 2D axisymmetric finite element simulations.

Approach 2: Solving inverse 3D indentation problems (e.g., with Berkovich tip) by integrating 2D axisymmetric FEM data (low fidelity) with 3D FEM data (high fidelity)

Traditionally, algorithms based on conical indentation finite element results were used for obtaining approximate solutions of Vickers or Berkovich 3D indentation problems [50, 46, 121, 28, 31, 220]. Here, we integrate the low-cost 2D axisymmetric finite element data (low fidelity) with a limited number of 3D finite element simulation data (high fidelity) to solve the Berkovich indentation inverse problem.

Approach 3: Learning and correcting material and/or setup specific systematic errors by including a few experimental data as part of the high-fidelity training data

In instrumented-indentation experiments, material-specific (e.g., for a material that is not well-represented by power-law hardening) and/or equipment-specific (e.g., nonlinear machine compliance) systematic errors can be significantly en-

larged when performing inverse analyses. We attempt to overcome this issue by adding a few experimental data as part of the high-fidelity training data in Approach 2. Specifically, the experimental data added for training can come from the same material using the same experimental setup or from a different calibration material tested under the same experimental conditions.

4.4.5 Data availability

The code and related input data have been deposited in GitHub at https://github.com/lululxvi/deep-learning-for-indentation. All other data are included in the manuscript and Appendix C.

Chapter

FIVE

Deep Learning for Solving Differential Equations

Manuscript information

A version of this chapter appeared in [136]: L. Lu, X. Meng, Z. Mao, & G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

Author contributions: L.L. designed the project and implemented the software. X.M. and Z.M developed the residual-based adaptive refinement. L.L., X.M., and Z.M. carried out the numerical examples. L.L., X.M., Z.M., and G.E.K. wrote the manuscript. G.E.K. supervised the work.

5.1 Introduction

In the last 15 years, deep learning in the form of deep neural networks (NNs), has been used very effectively in diverse applications [123], such as computer vision and natural language processing. Despite the remarkable success in these and related areas, deep learning has not yet been widely used in the field of scientific computing. However, more recently, solving partial differential equations (PDEs), e.g., in the standard differential form or in the integral form, via deep learning has emerged as a potentially new sub-field under the name of Scientific Machine Learning (SciML) [12]. In particular, we can replace traditional numerical discretization methods with a neural network that approximates the solution to a PDE.

To obtain the approximate solution of a PDE via deep learning, a key step is

to constrain the neural network to minimize the PDE residual, and several approaches have been proposed to accomplish this. Compared to the traditional mesh-based methods, such as the finite difference method (FDM) and the finite element method (FEM), deep learning could be a mesh-free approach by taking advantage of the automatic differentiation [181], and could break the curse of dimensionality [176, 75]. Among these approaches, some can only be applied to particular types of problems, such as image-like input domain [111, 133, 243] or parabolic PDEs [18, 79]. Some researchers adopt the variational form of PDEs and minimize the corresponding energy functional [61, 85]. However, not all PDEs can be derived from a known functional, and thus Galerkin type projections have also been considered [144]. Alternatively, one could use the PDE in strong form directly [53, 218, 119, 120, 20, 196, 181]; in this form, automatic differentiation could be used directly to avoid truncation errors and the numerical quadrature errors of variational forms. This strong form approach was introduced in [181] coining the term physics-informed neural networks (PINNs). An attractive feature of PINNs is that it can be used to solve inverse problems with minimum change of the code for forward problems [181, 182, 212, 88, 40]. In addition, PINNs have been further extended to solve integro-differential equations (IDEs), fractional differential equations (FDEs) [170], and stochastic differential equations (SDEs) [235, 227, 151, 234]. However, unlike traditional numerical methods, PINN solutions are obtained by solving highly nonlinear and non-convex optimization problems. In practice, to achieve a good level of accuracy, we need to tune all the hyperparameters, e.g., network size and learning rate. We also note that PINNs may converge to different solutions from different network initial values.

In this chapter, we present various PINN algorithms implemented in a Python library DeepXDE[1], which is designed to serve both as an education tool to be

[1]Source code is published under the Apache License, Version 2.0 on GitHub. https://github.

used in the classroom as well as a research tool for solving problems in computational science and engineering (CSE). DeepXDE can be used to solve multi-physics problems, and supports complex-geometry domains based on the technique of constructive solid geometry (CSG), hence avoiding tedious and time-consuming computational geometry tasks. By using DeepXDE, from the implementation point of view, time-dependent PDEs can be solved as easily as steady states by only defining the initial conditions. In addition to the main workflow of DeepXDE, users can readily monitor and modify the solution process via `callback` functions, e.g., monitoring the Fourier spectrum of the neural network solution, which can reveal the learning mode of the NN Fig. 5.2. Last but not least, DeepXDE is designed to make the user code stay compact and manageable, resembling closely the mathematical formulation.

The chapter is organized as follows. In Section 5.2, after briefly introducing deep neural networks and automatic differentiation, we present the algorithm, approximation theory, and error analysis of PINNs, and make a comparison between PINNs and FEM. We then discuss how to use PINNs to solve integro-differential equations and inverse problems. In addition, we propose the residual-based adaptive refinement (RAR) method to improve the training efficiency of PINNs. In Section 5.3, we introduce the usage of our library, DeepXDE, and its customizability. In Section 5.4, we demonstrate the capability of PINNs and friendly use of DeepXDE for five different examples. Finally, we conclude the chapter in Section 5.5.

`com/lululxvi/deepxde`

5.2 Algorithm and theory of physics-informed neural networks

In this section, we first provide a brief overview of deep neural networks and automatic differentiation, and present the algorithm and theory of PINNs for solving PDEs. We then make a comparison between PINNs and FEM, and discuss how to use PINNs to solve integro-differential equations and inverse problems. Next we propose RAR, an efficient way to select the residual points adaptively during the training process.

5.2.1 Deep neural networks

Mathematically, a deep neural network is a particular choice of a compositional function. The simplest neural network is the feed-forward neural network (FNN), also called multilayer perceptron (MLP), which applies linear and nonlinear transformations to the inputs recursively. Although many different types of neural networks have been developed in the past decades, such as the convolutional neural network and the recurrent neural network. In this chapter we consider FNN, which is sufficient for most PDE problems, and residual neural network (ResNet), which is easier to train for deep networks. However, it is straightforward to employ other types of neural networks.

Let $\mathcal{N}^L(\mathbf{x}) : \mathbb{R}^{d_{\text{in}}} \to \mathbb{R}^{d_{\text{out}}}$ be a L-layer neural network, or a $(L-1)$-hidden layer neural network, with N_ℓ neurons in the ℓ-th layer ($N_0 = d_{\text{in}}$, $N_L = d_{\text{out}}$). Let us denote the weight matrix and bias vector in the ℓ-th layer by $\boldsymbol{W}^\ell \in \mathbb{R}^{N_\ell \times N_{\ell-1}}$ and $\mathbf{b}^\ell \in \mathbb{R}^{N_\ell}$, respectively. Given a nonlinear activation function σ, which is applied

element-wisely, the FNN is recursively defined as follows:

$$\begin{aligned}
\text{input layer:} \quad & \mathcal{N}^0(\mathbf{x}) = \mathbf{x} \in \mathbb{R}^{d_{\text{in}}}, \\
\text{hidden layers:} \quad & \mathcal{N}^\ell(\mathbf{x}) = \sigma(\boldsymbol{W}^\ell \mathcal{N}^{\ell-1}(\mathbf{x}) + \boldsymbol{b}^\ell) \in \mathbb{R}^{N_\ell}, \quad \text{for} \quad 1 \le \ell \le L-1, \\
\text{output layer:} \quad & \mathcal{N}^L(\mathbf{x}) = \boldsymbol{W}^L \mathcal{N}^{L-1}(\mathbf{x}) + \boldsymbol{b}^L \in \mathbb{R}^{d_{\text{out}}};
\end{aligned}$$

see also a visualization of a neural network in Fig. 5.1. Commonly used activation functions include the logistic sigmoid $1/(1+e^{-x})$, the hyperbolic tangent (tanh), and the rectified linear unit (ReLU, $\max\{x, 0\}$).

5.2.2 Automatic differentiation

In PINNs, it is required to compute the derivatives of the network outputs with respect to the network inputs. There are four possible methods for computing the derivatives [16, 143]: (1) hand-coded analytical derivative; (2) finite difference or other numerical approximations; (3) symbolic differentiation (used in software programs such as Mathematica, Maxima, and Maple); and (4) automatic differentiation (AD, also called algorithmic differentiation). In deep learning, the derivatives are evaluated using backpropagation [186], a specialized technique of AD.

Considering the fact that the neural network represents a compositional function, then AD applies the chain rule repeatedly to compute the derivatives. There are two steps in AD: one forward pass to compute the values of all variables, and one backward pass to compute the derivatives. To demonstrate AD, we consider

a FNN of only one hidden layer with two inputs x_1 and x_2 and one output y:

$$
\begin{aligned}
v &= -2x_1 + 3x_2 + 0.5, \\
h &= \tanh v, \\
y &= 2h - 1.
\end{aligned}
$$

The forward pass and backward pass of AD for computing the partial derivatives $\frac{\partial y}{\partial x_1}$ and $\frac{\partial y}{\partial x_2}$ at $(x_1, x_2) = (2, 1)$ are shown in Table 5.1.

We can see that AD only requires one forward pass and one backward pass to compute all the partial derivatives, no matter what the input dimension is. In contrast, using finite differences computing each partial derivative $\frac{\partial y}{\partial x_i}$ requires two function valuations $y(x_1, \ldots, x_i, \ldots, x_{d_{\text{in}}})$ and $y(x_1, \ldots, x_i + \Delta x_i, \ldots, x_{d_{\text{in}}})$ for some small number Δx_i, and thus in total $d_{\text{in}} + 1$ forward passes are required to evaluate all the partial derivatives. Hence, AD is much more efficient than finite difference when the input dimension is high (see [16, 143] for more details of the comparison between AD and the other three methods). To compute n^{th}-order derivatives, AD can be applied recursively n times. However, this nested approach may lead to inefficiency and numerical instability, and hence other methods, e.g., Taylor-Mode AD, have been developed for this purpose [21, 22]. Finally we note that with AD we differentiate the NN and therefore we can deal with noisy data [170].

Table 5.1: Example of AD to compute the partial derivatives $\frac{\partial y}{\partial x_1}$ and $\frac{\partial y}{\partial x_2}$ at $(x_1, x_2) = (2, 1)$.

Forward pass	Backward pass
$x_1 = 2$ $x_2 = 1$	$\frac{\partial y}{\partial y} = 1$
$v = -2x_1 + 3x_2 + 0.5 = -0.5$ $h = \tanh v \approx -0.462$	$\frac{\partial y}{\partial h} = \frac{\partial (2h-1)}{\partial h} = 2$ $\frac{\partial y}{\partial v} = \frac{\partial y}{\partial h}\frac{\partial h}{\partial v} = \frac{\partial y}{\partial h}\text{sech}^2(v) \approx 1.573$
$y = 2h - 1 = -1.924$	$\frac{\partial y}{\partial x_1} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial x_1} = \frac{\partial y}{\partial v} \times (-2) = -3.146$ $\frac{\partial y}{\partial x_2} = \frac{\partial y}{\partial v}\frac{\partial v}{\partial x_2} = \frac{\partial y}{\partial v} \times 3 = 4.719$

5.2.3 Physics-informed neural networks (PINNs) for solving PDEs

We consider the following PDE parameterized by $\boldsymbol{\lambda}$ for the solution $u(\mathbf{x})$ with $\mathbf{x} = (x_1, \ldots, x_d)$ defined on a domain $\Omega \subset \mathbb{R}^d$:

$$f\left(\mathbf{x}; \frac{\partial u}{\partial x_1}, \ldots, \frac{\partial u}{\partial x_d}; \frac{\partial^2 u}{\partial x_1 \partial x_1}, \ldots, \frac{\partial^2 u}{\partial x_1 \partial x_d}; \ldots; \boldsymbol{\lambda}\right) = 0, \quad \mathbf{x} \in \Omega, \tag{5.1}$$

with suitable boundary conditions

$$\mathcal{B}(u, \mathbf{x}) = 0 \quad \text{on} \quad \partial\Omega,$$

where $\mathcal{B}(u, \mathbf{x})$ could be Dirichlet, Neumann, Robin, or periodic boundary conditions. For time-dependent problems, we consider time t as a special component of $\mathbf{x}$, and Ω contains the temporal domain. The initial condition can be simply treated as a special type of Dirichlet boundary condition on the spatio-temporal domain.

The algorithm of PINN [120, 181] is shown in Procedure 1, and visually in the schematic of Fig. 5.1 solving a diffusion equation $\frac{\partial u}{\partial t} = \lambda \frac{\partial^2 u}{\partial x^2}$ with mixed boundary conditions $u(x, t) = g_D(x, t)$ on $\Gamma_D \subset \partial\Omega$ and $\frac{\partial u}{\partial \mathbf{n}}(x, t) = g_R(u, x, t)$

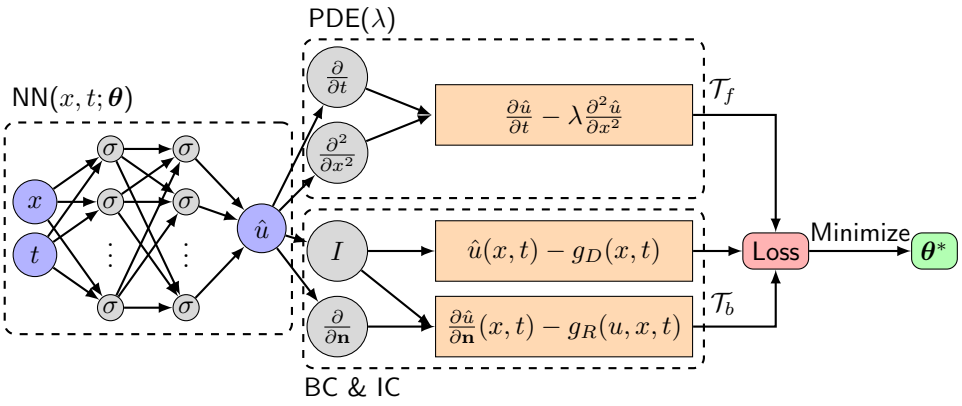


Figure 5.1: Schematic of a PINN for solving the diffusion equation $\frac{\partial u}{\partial t} = \lambda \frac{\partial^2 u}{\partial x^2}$ with mixed boundary conditions (BC) $u(x,t) = g_D(x,t)$ on $\Gamma_D \subset \partial\Omega$ and $\frac{\partial u}{\partial \mathbf{n}}(x,t) = g_R(u,x,t)$ on $\Gamma_R \subset \partial\Omega$. The initial condition (IC) is treated as a special type of boundary conditions. $\mathcal{T}_f$ and $\mathcal{T}_b$ denote the two sets of residual points for the equation and BC/IC.

Procedure 1: The PINN algorithm for solving differential equations.

Step 1 Construct a neural network $\hat{u}(\mathbf{x}; \boldsymbol{\theta})$ with parameters $\boldsymbol{\theta}$.

Step 2 Specify the two training sets $\mathcal{T}_f$ and $\mathcal{T}_b$ for the equation and boundary/initial conditions.

Step 3 Specify a loss function by summing the weighted L^2 norm of both the PDE equation and boundary condition residuals.

Step 4 Train the neural network to find the best parameters $\boldsymbol{\theta}^*$ by minimizing the loss function $\mathcal{L}(\boldsymbol{\theta}; \mathcal{T})$.

---

on $\Gamma_R \subset \partial\Omega$. We explain each step as follows. In a PINN, we first construct a neural network $\hat{u}(\mathbf{x};\boldsymbol{\theta})$ as a surrogate of the solution $u(\mathbf{x})$, which takes the input $\mathbf{x}$ and outputs a vector with the same dimension as u. Here, $\boldsymbol{\theta} = \{\boldsymbol{W}^\ell, \boldsymbol{b}^\ell\}_{1\le\ell\le L}$ is the set of all weight matrices and bias vectors in the neural network $\hat{u}$. One advantage of PINNs by choosing neural networks as the surrogate of u is that we can take the derivatives of $\hat{u}$ with respect to its input $\mathbf{x}$ by applying the chain rule for differentiating compositions of functions using the automatic differentiation (AD), which is conveniently integrated in machine learning packages, such as TensorFlow [1] and PyTorch [171].

In the next step, we need to restrict the neural network $\hat{u}$ to satisfy the physics imposed by the PDE and boundary conditions. In practice, we restrict $\hat{u}$ on some scattered points (e.g., randomly distributed points, or clustered points in the domain [142]), i.e., the training data $\mathcal{T} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{|\mathcal{T}|}\}$ of size $|\mathcal{T}|$. In addition, $\mathcal{T}$ is comprised of two sets $\mathcal{T}_f \subset \Omega$ and $\mathcal{T}_b \subset \partial\Omega$, which are the points in the domain and on the boundary, respectively. We refer $\mathcal{T}_f$ and $\mathcal{T}_b$ as the sets of "residual points".

To measure the discrepancy between the neural network $\hat{u}$ and the constraints, we consider the loss function defined as the weighted summation of the L^2 norm of residuals for the equation and boundary conditions:

$$\mathcal{L}(\boldsymbol{\theta};\mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta};\mathcal{T}_b), \tag{5.2}$$

where

$$\mathcal{L}_f(\boldsymbol{\theta};\mathcal{T}_f) = \frac{1}{|\mathcal{T}_f|}\sum_{\mathbf{x}\in\mathcal{T}_f}\left\|f\left(\mathbf{x};\frac{\partial\hat{u}}{\partial x_1},\ldots,\frac{\partial\hat{u}}{\partial x_d};\frac{\partial^2\hat{u}}{\partial x_1\partial x_1},\ldots,\frac{\partial^2\hat{u}}{\partial x_1\partial x_d};\ldots;\boldsymbol{\lambda}\right)\right\|_2^2,$$
$$\mathcal{L}_b(\boldsymbol{\theta};\mathcal{T}_b) = \frac{1}{|\mathcal{T}_b|}\sum_{\mathbf{x}\in\mathcal{T}_b}\|\mathcal{B}(\hat{u},\mathbf{x})\|_2^2,$$

and w_f and w_b are the weights. The loss involves derivatives, such as the partial derivative $\partial\hat{u}/\partial x_1$ or the normal derivative at the boundary $\partial\hat{u}/\partial\mathbf{n} = \nabla\hat{u}\cdot\mathbf{n}$, which are handled via AD.

In the last step, the procedure of searching for a good $\boldsymbol{\theta}$ by minimizing the loss $\mathcal{L}(\boldsymbol{\theta};\mathcal{T})$ is called "training". Considering the fact that the loss is highly nonlinear and non-convex with respect to $\boldsymbol{\theta}$ [23], we usually minimize the loss function by gradient-based optimizers, such as gradient descent, Adam [112], and L-BFGS [29]. We remark that based on our experience, for smooth PDE solutions L-BFGS can find a good solution with less iterations than Adam, because L-BFGS uses second-order derivatives of the loss function, while Adam only relies on first-order derivatives. However, for stiff solutions L-BFGS is more likely to be stuck at a bad local minimum. The required number of iterations highly depends on the problem (e.g., the smoothness of the solution), and to check whether the network converges or not, we can monitor the loss function or the PDE residual using `callback` functions. We also note that acceleration of training can be achieved by using adaptive activation function that may remove bad local minima, see [98, 97].

Unlike traditional numerical methods, for PINNs there is no guarantee of unique solutions, because PINN solutions are obtained by solving non-convex optimization problems, which in general do not have a unique solution. In practice, to achieve a good level of accuracy, we need to tune all the hyperparameters, e.g.,

network size, learning rate, and the number of residual points. The required network size depends highly on the smoothness of the PDE solution. For example, a small network (e.g., a few layers and twenty neurons per layer) is sufficient for solving the 1D Poisson equation, but a deeper and wider network is required for the 1D Burgers equation to achieve a similar level of accuracy. We also note that PINNs may converge to different solutions from different network initial values [170, 212, 88], and thus a common strategy is that we train PINNs from random initialization for a few times (e.g., 10 independent runs) and choose the network with the smallest training loss as the final solution.

In the algorithm of PINN introduced above, we enforce soft constraints of boundary/initial conditions through the loss $\mathcal{L}_b$. This approach can be used for complex domains and any type of boundary conditions. On the other hand, it is possible to enforce hard constraints for simple cases [119]. For example, when the boundary condition is $u(0) = u(1) = 0$ with $\Omega = [0, 1]$, we can simply choose the surrogate model as $\hat{u}(x) = x(x-1)\mathcal{N}(x)$ to satisfy the boundary condition automatically, where $\mathcal{N}(x)$ is a neural network.

We note that we have great flexibility in choosing the residual points $\mathcal{T}$, and here we provide three possible strategies:

1. We can specify the residual points at the beginning of training, which could be grid points on a lattice or random points, and never change them during the training process.

2. In each optimization iteration, we could select randomly different residual points.

3. We could improve the location of the residual points adaptively during

training, e.g., the method proposed in Section 5.2.8.

When the number of residual points required is very large, e.g., in multiscale problems, it is computationally expensive to calculate the loss and gradient in every iteration. Instead of using all residual points, we can split the residual points into small batches, and in each iteration we only use one batch to calculate the loss and update model parameters; this is the so-called "mini-batch" gradient descent. The aforementioned strategy (2), i.e., re-sampling in each step, is a special case of mini-batch gradient descent by choosing $\mathcal{T} = \Omega$ with $|\mathcal{T}| = \infty$.

Recent studies show that for function approximation, neural networks learn target functions from low to high frequencies [179, 226], but here we show that the learning mode of PINNs is different due to the existence of high-order derivatives. For example, when we approximate the function $f(x) = \sum_{k=1}^{5} \sin(2kx)/(2k)$ in $[-\pi, \pi]$ by a NN, the function is learned from low to high frequency (Fig. 5.2A). However, when we employ a PINN to solve the Poisson equation $-f_{xx} = \sum_{k=1}^{5} 2k \sin(2kx)$ with zero boundary conditions in the same domain, all frequencies are learned almost simultaneously (Fig. 5.2B). Interestingly, by comparing Fig. 5.2A and B we can see that at least in this case solving the PDE using a PINN is faster than approximating a function using a NN. We can monitor this training process using the `callback` functions in our library DeepXDE as discussed later.

5.2.4 Approximation theory and error analysis for PINNs

One fundamental question related to PINNs is whether there exists a neural network satisfying both the PDE equation and the boundary conditions, i.e., whether there exists a neural network that can simultaneously and uniformly approximate

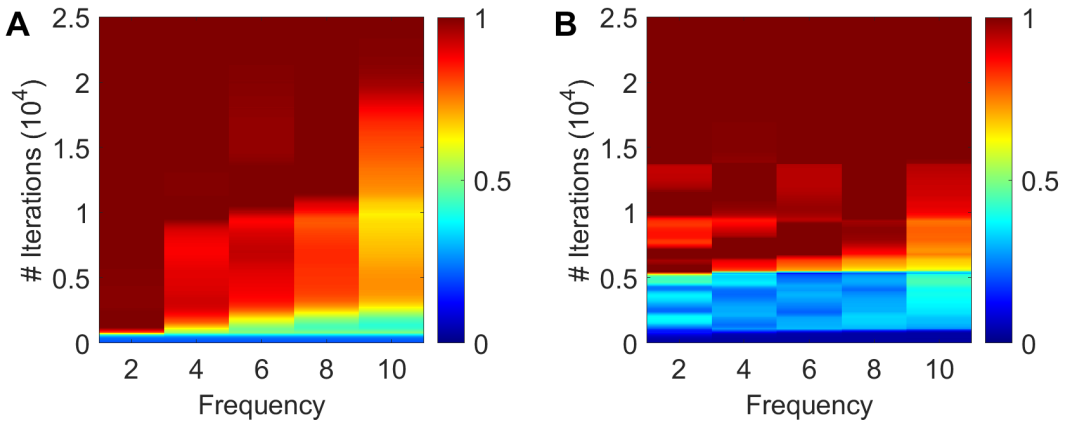


Figure 5.2: Convergence of the amplitude for each frequency during the training process. (**A**) A neural network is trained to approximate the function $f(x) = \sum_{k=1}^{5} \sin(2kx)/(2k)$. The color represents amplitude values with the maximum amplitude for each frequency normalized to 1. (**B**) A PINN is used to solve the Poisson equation $-f_{xx} = \sum_{k=1}^{5} 2k \sin(2kx)$ with zero boundary conditions. We use a neural network of 4 hidden layers and 20 neurons per layer. The learning rate is chosen as 10^{-4}, and 500 random points are sampled for training.

a function and its partial derivatives. To address this question, we first introduce some notation. Let $\mathbb{Z}_+^d$ be the set of d-dimensional nonnegative integers. For $\mathbf{m} = (m_1, \ldots, m_d) \in \mathbb{Z}_+^d$, we set $|\mathbf{m}| := m_1 + \cdots + m_d$, and

$$D^{\mathbf{m}} := \frac{\partial^{|\mathbf{m}|}}{\partial x_1^{m_1} \ldots \partial x_d^{m_d}}.$$

We say $f \in C^{\mathbf{m}}(\mathbb{R}^d)$ if $D^{\mathbf{k}} f \in C(\mathbb{R}^d)$ for all $\mathbf{k} \leq \mathbf{m}$, $\mathbf{k} \in \mathbb{Z}_+^d$, where $C(\mathbb{R}^d) = \{f : \mathbb{R}^d \to \mathbb{R} | f \text{ is continuous}\}$ is the space of continuous functions. Then, we recall the following theorem of derivative approximation using single hidden layer neural networks due to Pinkus [173].

Theorem 5.2.1. *Let $\mathbf{m}^i \in \mathbb{Z}_+^d$, $i = 1, \ldots, s$, and set $m = \max_{i=1,\ldots,s} |\mathbf{m}^i|$. Assume $\sigma \in C^m(\mathbb{R})$ and σ is not a polynomial. Then the space of single hidden layer neural nets*

$$\mathcal{M}(\sigma) := span\{\sigma(\mathbf{w} \cdot \mathbf{x} + b) : \mathbf{w} \in \mathbb{R}^d, b \in \mathbb{R}\}$$

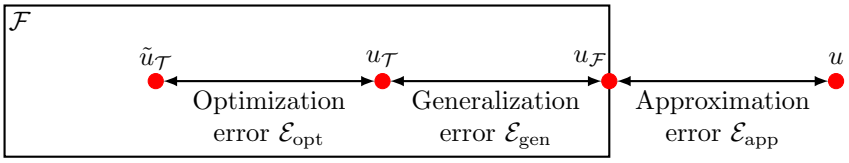


Figure 5.3: Illustration of errors of a PINN. The total error consists of the approximation error, the optimization error, and the generalization error. Here, u is the PDE solution, $u_\mathcal{F}$ is the best function close to u in the function space $\mathcal{F}$, $u_\mathcal{T}$ is the neural network whose loss is at a global minimum, and $\tilde{u}_\mathcal{T}$ is the function obtained by training a neural network.

is dense in

$$C^{\mathbf{m}^1,\dots,\mathbf{m}^s}(\mathbb{R}^d) := \cap_{i=1}^s C^{\mathbf{m}^i}(\mathbb{R}^d),$$

i.e., for any $f \in C^{\mathbf{m}^1,\dots,\mathbf{m}^s}(\mathbb{R}^d)$, *any compact* $K \subset \mathbb{R}^d$, *and any* $\varepsilon > 0$, *there exists a* $g \in \mathcal{M}(\sigma)$ *satisfying*

$$\max_{\mathbf{x}\in K} |D^{\mathbf{k}} f(\mathbf{x}) - D^{\mathbf{k}} g(\mathbf{x})| < \varepsilon,$$

for all $\mathbf{k} \in \mathbb{Z}_+^d$ *for which* $\mathbf{k} \le \mathbf{m}^i$ *for some* i.

Theorem 5.2.1 shows that feed-forward neural nets with enough neurons can simultaneously and uniformly approximate any function and its partial derivatives. However, neural networks in practice have limited size. Let $\mathcal{F}$ denote the family of all the functions that can be represented by our chosen neural network architecture. The solution u is unlikely to belong to the family $\mathcal{F}$, and we define $u_\mathcal{F} = \arg\min_{f\in\mathcal{F}} \|f - u\|$ as the best function in $\mathcal{F}$ close to u (Fig. 5.3). Because we only train the neural network on the training set $\mathcal{T}$, we define $u_\mathcal{T} = \arg\min_{f\in\mathcal{F}} \mathcal{L}(f;\mathcal{T})$ as the neural network whose loss is at global minimum. For simplicity, we assume that u, $u_\mathcal{F}$ and $u_\mathcal{T}$ are well defined and unique. Finding $u_\mathcal{T}$ by minimizing the loss is often computationally intractable [23], and our optimizer returns an approximate solution $\tilde{u}_\mathcal{T}$.

We can then decompose the total error $\mathcal{E}$ as [26]

$$\mathcal{E} := \|\tilde{u}_{\mathcal{T}} - u\| \leq \underbrace{\|\tilde{u}_{\mathcal{T}} - u_{\mathcal{T}}\|}_{\mathcal{E}_{\text{opt}}} + \underbrace{\|u_{\mathcal{T}} - u_{\mathcal{F}}\|}_{\mathcal{E}_{\text{gen}}} + \underbrace{\|u_{\mathcal{F}} - u\|}_{\mathcal{E}_{\text{app}}}.$$

The approximation error $\mathcal{E}_{\text{app}}$ measures how closely $u_{\mathcal{F}}$ can approximate u. The generalization error $\mathcal{E}_{\text{gen}}$ is determined by the number/locations of residual points in $\mathcal{T}$ and the capacity of the family $\mathcal{F}$. Neural networks of larger size have smaller approximation errors but could lead to higher generalization errors, which is called bias-variance tradeoff. Overfitting occurs when the generalization error dominates. In addition, the optimization error $\mathcal{E}_{\text{opt}}$ stems from the loss function complexity and the optimization setup, such as learning rate and number of iterations. However, currently there is no error estimation for PINNs yet, and even quantifying the three errors for supervised learning is still an open research problem [138, 137, 101].

5.2.5 Comparison between PINNs and FEM

To further explain the ideas of PINNs and to help those with the knowledge of FEM understand PINNs more easily, we make a comparison between PINNs and FEM point by point (Table 5.2):

- In FEM we approximate the solution u by a piecewise polynomial with point values to be determined, while in PINNs we construct a neural network as the surrogate model parameterized by weights and biases.
- FEM typically requires a mesh generation, while PINN is totally mesh-free, and we can use either a grid or random points.

- FEM converts a PDE to an algebraic system, using the stiffness matrix and mass matrix, while PINN embeds the PDE and boundary conditions into the loss function.
- In the last step, the algebraic system in FEM is solved exactly by a linear solver, but the network in PINN is learned by a gradient-based optimizer.

At a more fundamental level, PINNs provide a nonlinear approximation to the function and its derivatives whereas FEM represent a linear approximation.

Table 5.2: Comparison between PINN and FEM.

	PINN	FEM
Basis function	Neural network (nonlinear)	Piecewise polynomial (linear)
Parameters	Weights and biases	Point values
Training points	Scattered points (mesh-free)	Mesh points
PDE embedding	Loss function	Algebraic system
Parameter solver	Gradient-based optimizer	Linear solver
Errors	$\mathcal{E}_{\text{app}}$, $\mathcal{E}_{\text{gen}}$ and $\mathcal{E}_{\text{opt}}$ (Section 5.2.4)	Approximation/quadrature errors
Error bounds	Not available yet	Partially available [47, 102]

5.2.6 PINNs for solving integro-differential equations

When solving integro-differential equations (IDEs), we still employ the automatic differentiation technique to analytically derive the integer-order derivatives, while we approximate integral operators numerically using classical methods (Fig. 5.4) [170], such as Gaussian quadrature. Therefore, we introduce a fourth error component, the discretization error $\mathcal{E}_{\text{dis}}$, due to the approximation of the integral by Gaussian quadrature.

For example, when solving

$$\frac{dy}{dx} + y(x) = \int_0^x e^{t-x} y(t) dt,$$

we first use Gaussian quadrature of degree n to approximate the integral

$$\int_0^x e^{t-x} y(t) dt \approx \sum_{i=1}^{n} w_i e^{t_i(x)-x} y(t_i(x)),$$

and then we use a PINN to solve the following PDE instead of the original equation

$$\frac{dy}{dx} + y(x) \approx \sum_{i=1}^{n} w_i e^{t_i(x)-x} y(t_i(x)).$$

PINNs can also be easily extended to solve FDEs [170] and SDEs [235, 227, 151, 234], but we do not discuss here such cases due to the page limit.

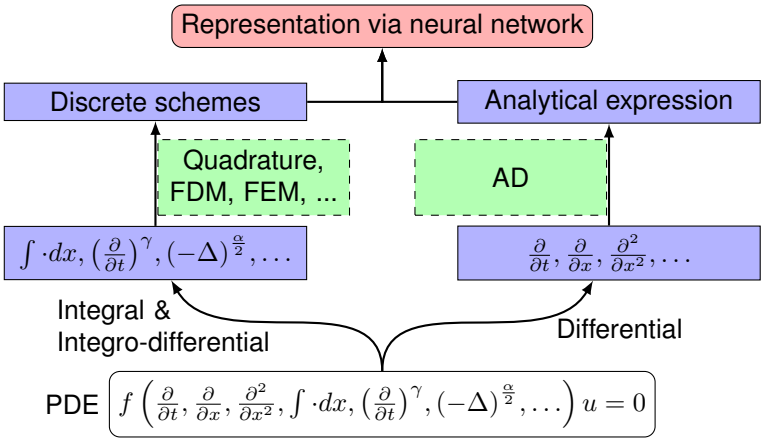


Figure 5.4: Schematic illustrating the modificaiton of the PINN algorithm for solving integro-differential equations. We employ the automatic differentiation technique to analytically derive the integer-order derivatives, and we approximate integral operators numerically using standard methods. (The figure is reproduced from [170].)

5.2.7 PINNs for solving inverse problems

In inverse problems, there are some unknown parameters $\boldsymbol{\lambda}$ in Eq. (5.1), but we have some extra information on some points $\mathcal{T}_i \subset \Omega$ besides the differential equation and boundary conditions:

$$\mathcal{I}(u, \mathbf{x}) = 0, \quad \text{for} \quad \mathbf{x} \in \mathcal{T}_i.$$

From the implementation point of view, PINNs solve inverse problems as easily as forward problems [181, 170]. The only difference between solving forward and inverse problems is that we add an extra loss term to Eq. (5.2):

$$\mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}) = w_f \mathcal{L}_f(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_f) + w_b \mathcal{L}_b(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_b) + w_i \mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i),$$

where

$$\mathcal{L}_i(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}_i) = \frac{1}{|\mathcal{T}_i|} \sum_{\mathbf{x} \in \mathcal{T}_i} \|\mathcal{I}(\hat{u}, \mathbf{x})\|_2^2.$$

We then optimize $\boldsymbol{\theta}$ and $\boldsymbol{\lambda}$ together, and our solution is

$$\boldsymbol{\theta}^*, \boldsymbol{\lambda}^* = \arg\min_{\boldsymbol{\theta}, \boldsymbol{\lambda}} \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{\lambda}; \mathcal{T}).$$

5.2.8 Residual-based adaptive refinement (RAR)

As we discussed in Section 5.2.3, the residual points $\mathcal{T}$ are usually randomly distributed in the domain. This works well for most cases, but it may not be efficient for certain PDEs that exhibit solutions with steep gradients. Take the Burgers equation as an example, intuitively we should put more points near the

sharp front to capture the discontinuity well. However, it is challenging, in general, to design a good distribution of residual points for problems whose solution is unknown. To overcome this challenge, we propose a residual-based adaptive refinement (RAR) method to improve the distribution of residual points during training process (Procedure 2), conceptually similar to FEM refinement methods [3]. The idea of RAR is that we will add more residual points in the locations where the PDE residual $\left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \dots; \boldsymbol{\lambda}\right)\right\|$ is large, and we repeat adding points until the mean residual

$$\mathcal{E}_r = \frac{1}{V} \int_\Omega \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \dots; \boldsymbol{\lambda}\right)\right\| d\mathbf{x} \tag{5.3}$$

is smaller than a threshold $\mathcal{E}_0$, where V is the volume of Ω.

Procedure 2: RAR for improving the distribution of residual points for training.

Step 1 Select the initial residual points $\mathcal{T}$, and train the neural network for a limited number of iterations.

Step 2 Estimate the mean PDE residual $\mathcal{E}_r$ in Eq. (5.3) by Monte Carlo integration, i.e., by the average of values at a set of randomly sampled dense locations $\mathcal{S} = \{\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_{|\mathcal{S}|}\}$:

$$\mathcal{E}_r \approx \frac{1}{|\mathcal{S}|} \sum_{\mathbf{x} \in \mathcal{S}} \left\| f\left(\mathbf{x}; \frac{\partial \hat{u}}{\partial x_1}, \dots, \frac{\partial \hat{u}}{\partial x_d}; \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_1}, \dots, \frac{\partial^2 \hat{u}}{\partial x_1 \partial x_d}; \dots; \boldsymbol{\lambda}\right)\right\|.$$

Step 3 Stop if $\mathcal{E}_r < \mathcal{E}_0$. Otherwise, add m new points with the largest residuals in $\mathcal{S}$ to $\mathcal{T}$, retrain the network, and then go to Step 2.

5.3 DeepXDE usage and customization

In this section, we introduce the usage of DeepXDE and how to customize DeepXDE to meet new problem requirements.

5.3.1 Usage

Compared to traditional numerical methods, the code written with DeepXDE is much shorter and more comprehensive, resembling closely the mathematical formulation. Solving differential equations in DeepXDE is no more than specifying the problem using the build-in modules, including computational domain (geometry and time), PDE equations, boundary/initial conditions, constraints, training data, neural network architecture, and training hyperparameters. The workflow is shown in Procedure 3 and Fig. 5.5.

Procedure 3: Usage of DeepXDE for solving differential equations.

Step 1 Specify the computational domain using the **`geometry`** module.

Step 2 Specify the PDE using the grammar of **`TensorFlow`**.

Step 3 Specify the boundary and initial conditions.

Step 4 Combine the geometry, PDE and boundary/initial conditions together into **`data.PDE`** or **`data.TimePDE`** for time-independent problems or for time-dependent problems, respectively. To specify training data, we can either set the specific point locations, or only set the number of points and then DeepXDE will sample the required number of points on a grid or randomly.

Step 5 Construct a neural network using the **`maps`** module.

Step 6 Define a **`Model`** by combining the PDE problem in Step 4 and the neural net in Step 5.

Step 7 Call **`Model.compile`** to set the optimization hyperparameters, such as optimizer and learning rate. The weights in Eq. (5.2) can be set here by **`loss_weights`**.

Step 8 Call **`Model.train`** to train the network from random initialization or a pre-trained model using the argument **`model_restore_path`**. It is extremely flexible to monitor and modify the training behavior using **`callbacks`**.

Step 9 Call **`Model.predict`** to predict the PDE solution at different locations.

In DeepXDE, the built-in primitive geometries include **`interval`**, **`triangle`**,

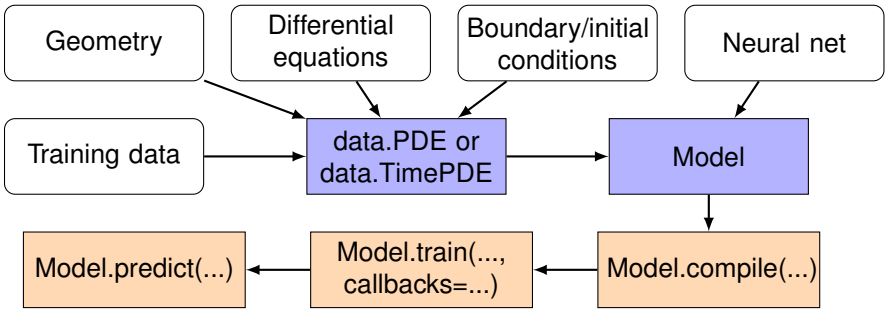


Figure 5.5: Flowchart of DeepXDE corresponding to Procedure 3. The white boxes define the PDE problem and the training hyperparameters. The blue boxes combine the PDE problem and training hyperparameters in the white boxes. The orange boxes are the three steps (from right to left) to solve the PDE.

`rectangle`, **`polygon`**, **`disk`**, **`cuboid`** and **`sphere`**. Other geometries can be constructed from these primitive geometries using three boolean operations: **`union (|)`**, **`difference (-)`** and **`intersection (&)`**. This technique is called constructive solid geometry (CSG), see Fig. 5.6 for examples. CSG supports both two-dimensional and three-dimensional geometries.

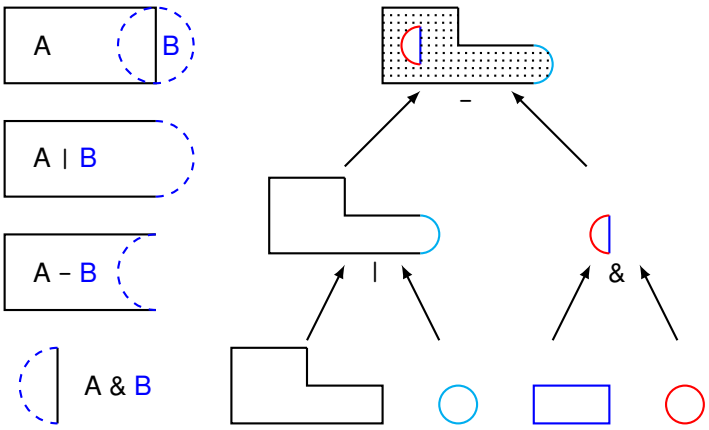


Figure 5.6: Examples of constructive solid geometry (CSG) in 2D. (**left**) A and B represent the rectangle and circle, respectively. The union $A|B$, difference $A - B$, and intersection $A\&B$ are constructed from A and B. (**right**) A complex geometry (top) is constructed from a polygon, a rectangle and two circles (bottom) through the union, difference, and intersection operations. This capability is included in the module `geometry` of DeepXDE.

DeepXDE supports four standard boundary conditions, including Dirichlet (**`DirichletBC`**), Neumann (**`NeumannBC`**), Robin (**`RobinBC`**), and periodic (**`PeriodicBC`**), and a more general BC can be defined using **`OperatorBC`**. The initial condition can be defined using **`IC`**. There are two types of neural networks available in

DeepXDE: feed-forward neural network (**`maps.FNN`**) and residual neural network (**`maps.ResNet`**). It is also convenient to choose different training hyperparameters, such as loss types, metrics, optimizers, learning rate schedules, initializations and regularizations.

In addition to solving differential equations, DeepXDE can also be used to approximate functions from multi-fidelity data [146], and learn nonlinear operators [135].

5.3.2 Customizability

All the components of DeepXDE are loosely coupled, and thus DeepXDE is well-structured and highly configurable. In this subsection, we discuss how to customize DeepXDE to address new problem requirements, e.g., new geometry or network architecture.

Geometry

As we introduced above, DeepXDE has already supported 7 basic geometries and the CSG technique. However, it is still possible that the user needs a new geometry, which cannot be constructed in DeepXDE. In this situation, a new geometry can be defined as shown in Procedure 4. Currently DeepXDE does not support accurately descriptions of complex curvilinear boundaries; however, a future extension could be incorporation of the non-uniform rational basis spline (NURBS) [94] for such representations.

Procedure 4: Customization of the new geometry module **`MyGeometry`**. The class methods should only be implemented as needed.

```
class MyGeometry(Geometry):
    def inside(self, x):
        """Check if x is inside the geometry."""
    def on_boundary(self, x):
        """Check if x is on the geometry boundary."""
    def boundary_normal(self, x):
        """Compute the unit normal at x for Neumann or Robin boundary conditions.
    """
    def periodic_point(self, x, component):
        """Compute the periodic image of x for periodic boundary condition."""
    def uniform_points(self, n, boundary=True):
        """Compute the equispaced point locations in the geometry."""
    def random_points(self, n, random="pseudo"):
        """Compute the random point locations in the geometry."""
    def uniform_boundary_points(self, n):
        """Compute the equispaced point locations on the boundary."""
    def random_boundary_points(self, n, random="pseudo"):
        """Compute the random point locations on the boundary."""
```

Neural networks

DeepXDE currently supports two neural networks: feed-forward neural network (**`maps.FNN`**) and residual neural network (**`maps.ResNet`**). A new network can be added as shown in Procedure 5.

Procedure 5: Customization of the neural network **`MyNet`**.

```
class MyNet(Map):
    @property
    def inputs(self):
        """Return the net inputs."""
    @property
    def outputs(self):
        """Return the net outputs."""
    @property
    def targets(self):
        """Return the targets of the net outputs."""
    def build(self):
        """Construct the network."""
```

Callbacks

It is usually a good strategy to monitor the training process of the neural network, and then make modifications in real time, e.g., change the learning rate. In

DeepXDE, this can be implemented by adding a callback function, and here we only list a few commonly used ones already implemented in DeepXDE:

- **`ModelCheckpoint`**, which saves the model after certain epochs or when a better model is found.
- **`OperatorPredictor`**, which calculates the values of the operator applying on the outputs.
- **`FirstDerivative`**, which calculates the first derivative of the outpus with respect to the inputs. This is a special case of **`OperatorPredictor`** with the operator being the first derivative.
- **`MovieDumper`**, which dumps the movie of the function during the training progress, and/or the movie of the spectrum of its Fourier transform.

It is very convenient to add other **`callback`** functions, which will be called at different stages of the training process, see Procedure 6.

Procedure 6: Customization of the callback **`MyCallback`**. Here, we only show how to add functions to be called at the beginning/end of every epoch. Similarly, we can call functions at the other training stages, such as at the beginning of training.

```
class MyCallback(Callback):
    def on_epoch_begin(self):
        """Called at the beginning of every epoch."""
    def on_epoch_end(self):
        """Called at the end of every epoch."""
```

5.4 Demonstration examples

In this section, we use PINNs and DeepXDE to solve different problems. In all examples, we use the tanh as the activation function, and the other hyperparameters are listed in Table 5.3. The weights w_f, w_b and w_i in the loss function are set as 1. The codes of all examples are published in GitHub.

Table 5.3: Hyperparameters used for the following 5 examples. "Adam, L-BFGS" represents that we first use Adam for a certain number of iterations, and then switch to L-BFGS. The optimizer L-BFGS does not require learning rate, and the neural network is trained until convergence, so the number of iterations is also ignored for L-BFGS.

Example	NN Depth	NN Width	Optimizer	Learning rate	# Iterations
1	4	50	Adam, L-BFGS	0.001	50000
2	3	20	Adam, L-BFGS	0.001	10000
3	3	40	Adam	0.001	60000
4	3	20	Adam	0.001	80000
5	4	20	L-BFGS	-	-

5.4.1 Poisson equation over an L-shaped domain

Consider the following two-dimensional Poisson equation over an L-shaped domain $\Omega = [-1, 1]^2 \setminus [0, 1]^2$:

$$\begin{aligned} -\Delta u(x, y) &= 1, \quad (x, y) \in \Omega, \\ u(x, y) &= 0, \quad (x, y) \in \partial\Omega. \end{aligned}$$

We choose 1200 and 120 random points drawn from a uniform distribution as $\mathcal{T}_f$ and $\mathcal{T}_b$, respectively. The PINN solution is given in Fig. 5.7B. For comparison, we also present the numerical solution obtained by using the spectral element method

(SEM) [105] (Fig. 5.7A). The result of the absolute error is shown in Fig. 5.7C.

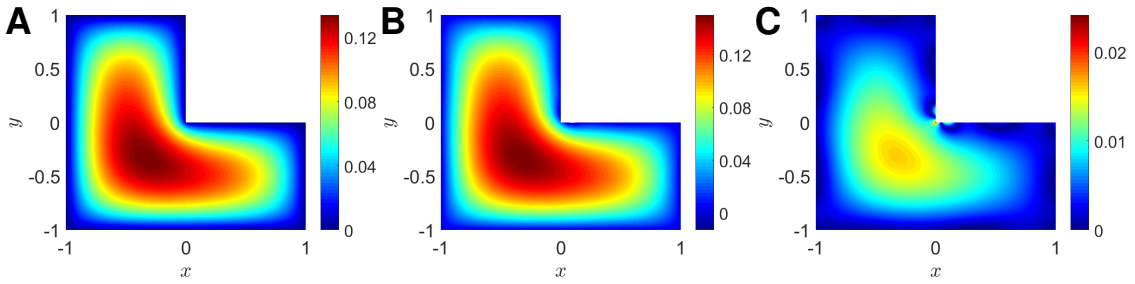


Figure 5.7: Example 5.4.1. Comparison of the PINN solution with the solution obtained by using spectral element method (SEM). (**A**) the SEM solution u_{SEM}, (**B**) the PINN solution u_{NN}, (**C**) the absolute error $|u_{SEM} - u_{NN}|$.

5.4.2 RAR for Burgers equation

We first consider the 1D Burgers equation:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu\frac{\partial^2 u}{\partial x^2}, \quad x \in [-1, 1],\ t \in [0, 1],$$
$$u(x, 0) = -\sin(\pi x), \quad u(-1, t) = u(1, t) = 0.$$

Let $\nu = 0.01/\pi$. Initially, we randomly select 2500 points (spatio-temporal domain) as the residual points, and then 40 more residual points are added adaptively via RAR developed in Section 5.2.8 with $m = 1$, $|\mathcal{S}| = 100000$ and $\mathcal{E}_0 = 0.005$. The training procedure after adding new points is shown in Table 5.3. We compare the PINN solution with RAR and the PINN solution based on 2540 randomly selected training data (Fig. 5.8 A and B), and demonstrate that PINN with RAR can capture the discontinuity much better. For a comparison, the finite difference solutions using central difference scheme for space discretization and forward Euler scheme for time discretization for the Burgers equation in the conservative form are also shown in Fig. 5.8A. Here, we also present two examples for the residual points added via the RAR method. As shown in Fig. 5.8 C and D, the added points (green

crosses) are quite close to the sharp interface, which indicates the effectiveness of RAR.

We further solve the following two-dimensional Burgers equation using the RAR:

$$\begin{aligned}
\partial_t u + u\partial_x u + v\partial_y u &= \frac{1}{\mathrm{Re}}(\partial_x^2 u + \partial_y^2 u),\\
\partial_t v + u\partial_x v + v\partial_y v &= \frac{1}{\mathrm{Re}}(\partial_x^2 v + \partial_y^2 v),\\
x, y \in [0,1], \text{ and } t &\in [0,1],
\end{aligned}$$

where u and v are the velocities along the x and y directions, respectively. In addition, Re is a non-dimensional number (Reynolds number) defined as $\mathrm{Re} = UL/\nu$, in which U and L are respectively the characteristic velocity and length, and ν is the kinematic viscosity of fluid. The exact solution can be obtained [11] as

$$\begin{aligned}
u(x,y,t) &= \frac{3}{4} - \frac{1}{4\left[1+\exp((-4x+4y-t)\mathrm{Re}/32)\right]},\\
v(x,y,t) &= \frac{3}{4} + \frac{1}{4\left[1+\exp((-4x+4y-t)\mathrm{Re}/32)\right]},
\end{aligned}$$

using the Dirichlet boundary conditions on all boundaries. In the present study, Re is set to be 5000, which is quite challenging due to the fact that the high Reynolds number leads to steep gradient in the solution. Initially, 200 residual points are randomly sampled in the spatio-temporal domain, and 5000 and 1000 random points are used for each initial and boundary conditions, respectively. We only add 10 extra residual points using RAR, and the total number of the residual points is 210 after convergence. For comparison, we also test the case without the RAR using 210 randomly sampled residual points. The results are displayed in Fig. 5.8 E and F, demonstrating the effectiveness of RAR.

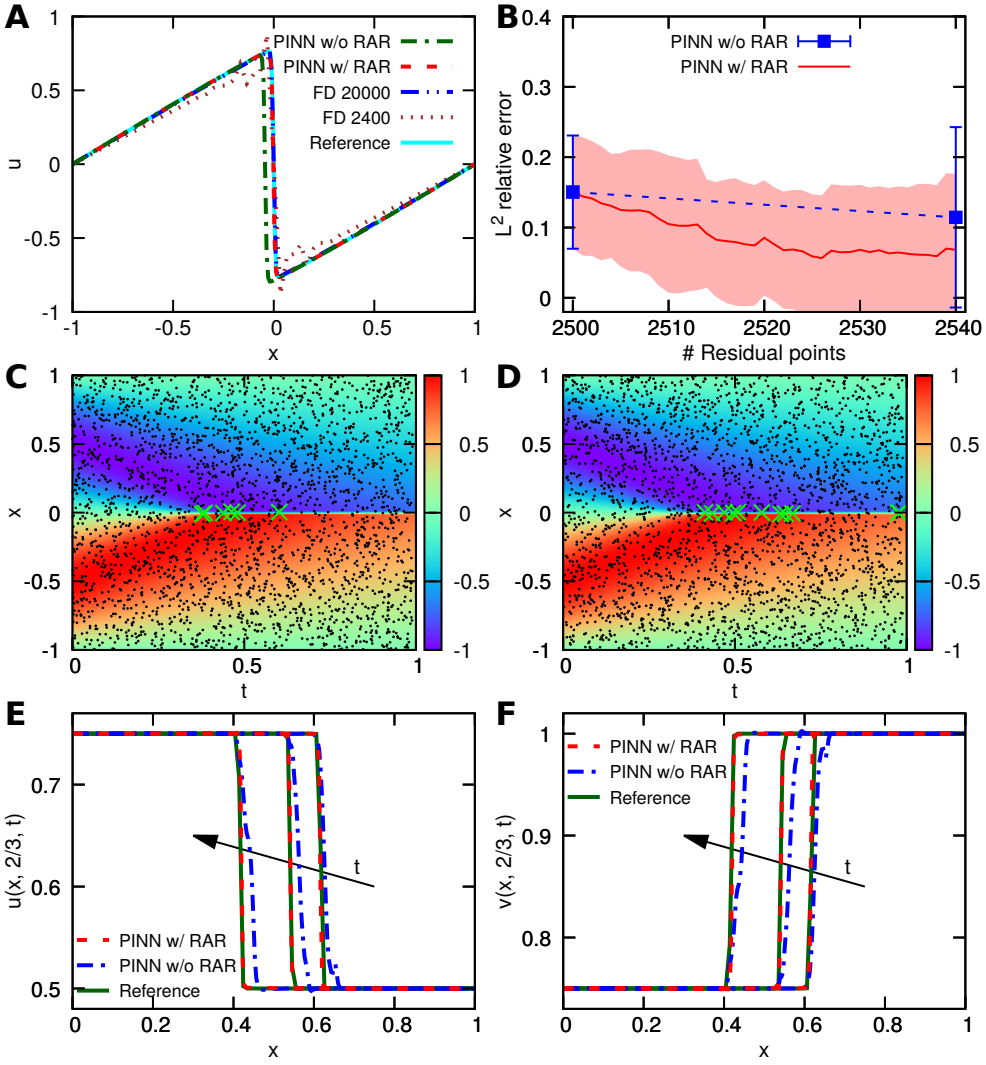


Figure 5.8: Example 5.4.2. Comparisons of the PINN solutions of (A, B, C and D) 1D and (E and F) 2D Burgers equations with and without RAR. (**A**) The cyan, green and red lines represent the reference solution of u from [181], PINN solution without RAR, PINN solution with RAR at $t = 0.9$, respectively. For the finite difference (FD) method, $200 \times 100 = 20000$ spatial-temporal grid points are used to achieve a good solution (blue line). If only $60 \times 40 = 2400$ points are used, the FD solution has large oscillations around the discontinuity (brown line). (**B**) L^2 relative error versus the number of residual points. The red solid line and shaded region correspond to the mean and one-standard-deviation band for the L^2 relative error of PINN with RAR, respectively. The blue dashed line is the mean and one-standard-deviation for the error of PINN using 2540 random residual points. The mean and standard deviation are obtained from 10 runs with random initial residual points. (**C** and **D**) Two representative examples for the residual points added via RAR. Black dots: initial residual points; green cross: added residual points; 6 and 11 residual points are added in C and D, respectively. (**E** and **F**) Comparison of the velocity profiles at $y = \frac{2}{3}$ from the PINNs with and without RAR for the 2D Burgers equation. The profiles at three different times ($t = 0.2$, 0.5 and 1) are presented with t increasing along the direction of the arrow.

5.4.3 Inverse problem for the Lorenz system

Consider the parameter identification problem of the following Lorenz system

$$\frac{dx}{dt} = \rho(y - x), \quad \frac{dy}{dt} = x(\sigma - z) - y, \quad \frac{dz}{dt} = xy - \beta z,$$

with the initial condition $(x(0), y(0), z(0)) = (-8, 7, 27)$, where ρ, σ and β are the three parameters to be identified from the observations at certain times. The observations are produced by solving the above system to $t = 3$ using Runge-Kutta (4,5) with the underlying true parameters $(\rho, \sigma, \beta) = (10, 15, 8/3)$. We choose 400 uniformly distributed random points and 25 equispaced points as the residual points $\mathcal{T}_f$ and $\mathcal{T}_i$, respectively. The evolution trajectories of ρ, σ and β are presented in Fig. 5.9A, with the final identified values of $(\rho, \sigma, \beta) = (10.002, 14.999, 2.668)$.

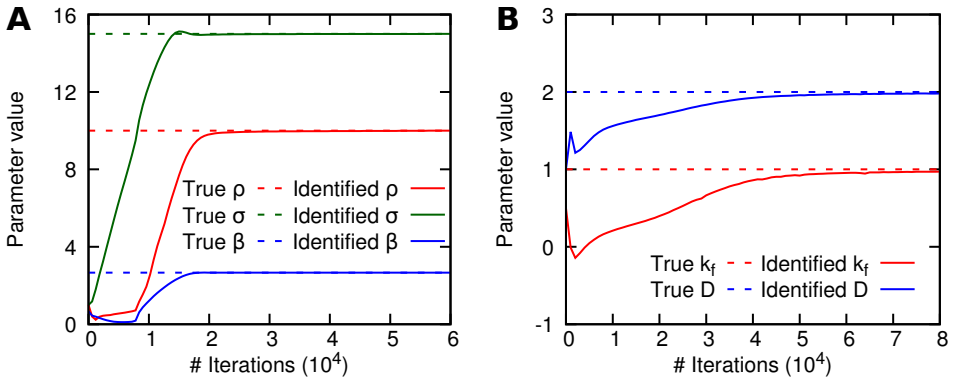


Figure 5.9: Examples 5.4.3 and 5.4.4. The identified values of (**A**) the Lorenz system and (**B**) diffusion-reaction system converge to the true values during the training process. The parameter values are scaled for plotting.

5.4.4 Inverse problem for diffusion-reaction systems

A diffusion-reaction system in porous media for the solute concentrations C_A, C_B and C_C ($A + 2B \rightarrow C$) is described by

$$\frac{\partial C_A}{\partial t} = D\frac{\partial^2 C_A}{\partial x^2} - k_f C_A C_B^2, \quad \frac{\partial C_B}{\partial t} = D\frac{\partial^2 C_B}{\partial x^2} - 2k_f C_A C_B^2, \quad x \in [0,1], t \in [0,10],$$
$$C_A(x,0) = C_B(x,0) = e^{-20x}, \quad C_A(0,t) = C_B(0,t) = 1, \quad C_A(1,t) = C_B(1,t) = 0,$$

where $D = 2 \times 10^{-3}$ is the effective diffusion coefficient, and $k_f = 0.1$ is the effective reaction rate. Because D and k_f depend on the pore structure and are difficult to measure directly, we estimate D and k_f based on 40000 observations of the concentrations C_A and C_B in the spatio-temporal domain. The identified D (1.98×10^{-3}) and k_f (0.0971) are displayed in Fig. 5.9B, which agree well with their true values.

5.4.5 Volterra IDE

Here, we consider the first-order integro-differential equation of the Volterra type in the domain $[0, 5]$:

$$\frac{dy}{dx} + y(x) = \int_0^x e^{t-x} y(t) dt, \quad y(0) = 1,$$

with the exact solution $y(x) = e^{-x} \cosh x$. We solve this IDE using the method in Section 5.2.6, and approximate the integral using Gaussian-Legendre quadrature of degree 20. The L^2 relative error is 2×10^{-3}, and the solution is shown in Fig. 5.10.

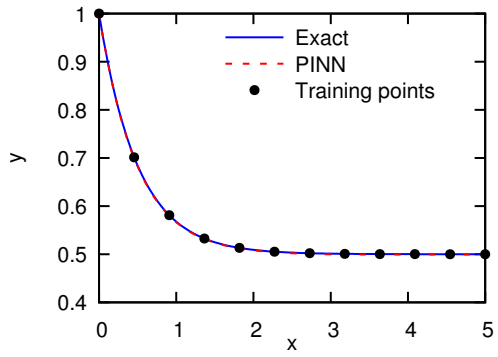


Figure 5.10: Example 5.4.5. The PINN algorithm for solving Volterra IDE. The blue solid line is the exact solution, and the red dash line is the numerical solution from PINN. 12 equispaced residual points (black dots) are used.

5.5 Concluding remarks

In this chapter, we present the algorithm, approximation theory, and error analysis of the physics-informed neural networks (PINNs) for solving different types of partial differential equations (PDEs). Compared to the traditional numerical methods, PINNs employ automatic differentiation to handle differential operators, and thus they are mesh-free. Unlike numerical differentiation, automatic differentiation does not differentiate the data and hence it can tolerate noisy data for training. We also discuss how to extend PINNs to solve other types of differential equations, such as integro-differential equations, and also how to solve inverse problems. In addition, we propose a residual-based adaptive refinement (RAR) method to improve the distribution of residual points during the training process, and thus increase the training efficiency.

To benefit both the education and the computational science communities, we have developed the Python library DeepXDE, an implementation of PINNs. By introducing the usage of DeepXDE, we show that DeepXDE enables user codes to be compact and follow closely the mathematical formulation. We also demonstrate

how to customize DeepXDE to meet new problem requirements. Our numerical examples for forward and inverse problems verify the effectiveness of PINNs and the capability of DeepXDE. Scientific machine learning is emerging as a new and potentially powerful alternative to classical scientific computing, so we hope that libraries such as DeepXDE will accelerate this development and will make it accessible to the classroom but also to other researchers who may find the need to adopt PINN-like methods in their research, which can be very effective especially for inverse problems.

Despite the aforementioned advantages, PINNs still have some limitations. For forward problems, PINNs are currently slower than finite elements but this can be alleviated via offline training [243, 223]. For long time integration, one can also use time-parallel methods to simultaneously compute on multiple GPUs for shorter time domains [147]. Another limitation is the search for effective neural network architectures, which is currently done empirically by users; however, emerging meta-learning techniques can be used to automate this search, see [244, 64]. Moreover, while here we enforce the strong form of PDEs, which is easy to be implemented by automatic differentiation, alternative weak/variational forms may also be effective, although they require the use of quadrature grids.

Chapter

SIX

DeepONet: Learning Nonlinear Operators based on the Universal Approximation Theorem of Operators

Manuscript information

The first version of this chapter appeared in [135]: L. Lu, P. Jin, & G. E. Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193v1*, 2019.

The current version of this chapter appeared in: L. Lu, P. Jin, G. Pang, Z. Zhang, & G. E. Karniadakis. DeepONet: Learning nonlinear operators based on the universal approximation theorem of operators. *Submitted*, 2020.

Author contributions: L.L. and G.E.K. designed the study based on G.E.K's original idea. L.L. developed DeepONet architectures. L.L., P.J., and Z.Z. developed the theory. L.L. performed the experiments of the integral, nonlinear ODE, gravity pendulum, and stochastic ODE/PDE operators. L.L. and P.J. performed the experiments of the Legendre transform, diffusion-reaction, advection, and advection-diffusion PDEs. G.P. performed the experiments of fractional operators. L.L., P.J., G.P., Z.Z., and G.E.K. wrote the manuscript. G.E.K. supervised the project.

6.1 Introduction

Over the last three centuries, mathematical models based on calculus have been employed to both predict and interpret physical phenomena around us, establishing fundamental laws that generalize well. Reflecting on the 25 centuries of

mathematics, from the Pythagoreans, to Newton's law, and relativity theories or the uncertainty principle, new descriptions of physical phenomena have followed the mathematics prevalent at that time together with prior knowledge. While calculus has served science well, new fundamental developments are slow and have not established any governing principles in some fields, e.g., in the emerging field of social dynamics. In the era of artificial intelligence and data analytics, the question is then if there are any more expressive representations that can be employed to accelerate scientific discovery. Classical calculus and partial differential equations (PDEs) or their stochastic variants, taught diligently in colleges around the globe, have been effective in the era of the "pursuit of simplicity" [213], but they may be inefficient or even inadequate to represent complex inter-connected systems or systems of systems with non-local interactions [222]. There is nothing unique about the calculus we use currently, and a concrete example is offered in [76], where the same set of data can be represented with the same accuracy by either an integer-order three-term PDE (in the form of the familiar advection-diffusion system) or by an unfamiliar two-term fractional order PDE, hence using the more expressive fractional calculus for discovering a new more compact PDE for time-dependent advection-diffusion systems.

Here, we investigate the possibility of formulating an alternative *calculus-agnostic* simple and fast way of describing an operator and its dynamics implicitly based on a give data set. More broadly, with the anticipated frequent human-robot interactions in the not too distant future, we address the question on how to best educate the new generations on quantitative inference and how to endow new robots with the required intelligence. Following the current paradigm, one approach would be to teach robots calculus, i.e., one integral or differential operator at a time, and train them to synthesize and solve such PDEs at blazing speeds to

make quantitative decisions, sense their environment, and communicate with humans; however, this seems computationally prohibitive. An alternative direction is to train them using new means in learning nonlinear operators and functionals that compactly express very complex dynamics without the need for resorting to prior knowledge of 17th-century calculus. A similar paradigm for discovering new physics concepts with no prior knowledge was provided in [96] using NNs and representation learning.

Towards the aforementioned high level goal, we will need to develop new types of NNs that go beyond the universal function approximation and supervised data [49, 91], and approximate functionals and nonlinear operators instead. As a first step, we resort to a little known but powerful theorem, the *universal operator approximation theorem* [37], which states that a NN with a single hidden layer can approximate accurately any nonlinear continuous *functional* (a mapping from a space of functions into the real numbers) [35, 148, 184] or (nonlinear) operator (a mapping from a space of functions into another space of functions) [37, 36]. To wit, let G be an operator taking an input function u with $G(u)$ the corresponding output function. For any point y in the domain of $G(u)$, the output $G(u)(y)$ is a real number. Hence, the network takes inputs composed of two parts: u and y, and outputs $G(u)(y)$ (Fig. 6.1A). Although our goal is to learn operators, which take a function as input, we have to represent these input functions discretely, so that network approximations can be applied. We envision that in the future such functions will be represented by other NNs. Here, we explore different representations of the input function, with the simplest one based on the function values at a sufficient but finite number of locations $\{x_1, x_2, \ldots, x_m\}$, which we call "sensors" (Fig. 6.1A). There are other ways to represent a function, e.g. with spectral expansions or as an image, and we demonstrate these in the examples

below. In particular, we demonstrate that we can design NNs that can represent accurately both explicit known operators, e.g., integrals, transforms, fractional derivatives and fractional Laplacians, as well as implicit operators known from classical calculus such as nonlinear, deterministic and stochastic ordinary and partial differential equations. The proposed NNs generalizes well, so given unseen functions they predict the action of the operator or predict the solution of the dynamical system accurately.

6.2 Materials and Methods

Next, we state the theorem due to Chen & Chen [37] (see Section D.1 for more details), based on which we propose the new NN, and subsequently we present data generation and a new theorem that relates the number and type of data with the accuracy of the input functions.

Theorem 6.2.1 (Universal Approximation Theorem for Operator). *Suppose that σ is a continuous non-polynomial function, X is a Banach Space, $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ are two compact sets in X and $\mathbb{R}^d$, respectively, V is a compact set in $C(K_1)$, G is a nonlinear continuous operator, which maps V into $C(K_2)$. Then for any $\epsilon > 0$, there are positive integers n, p, m, constants $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \dots, n$, $k = 1, \dots, p$, $j = 1, \dots, m$, such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \underbrace{\sum_{i=1}^{n} c_i^k \sigma\left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right)}_{branch} \underbrace{\sigma(w_k \cdot y + \zeta_k)}_{trunk} \right| < \epsilon \tag{6.1}$$

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

This approximation theorem is indicative of the potential application of neural networks to learn nonlinear operators from data, i.e., similar to standard NN where we learn functions from data. However, this theorem does not inform us how to learn operators efficiently. The overall accuracy of NNs can be characterized by dividing the total error into three main types: *approximation*, *optimization*, and *generalization* errors [26, 138, 101, 137]. The universal approximation theorem only guarantees a small approximation error for a sufficiently large network, but it does not consider the important optimization and generalization errors at all, which are often dominant contributions to the total error in practice. Useful networks should be easy to train, i.e., to exhibit small optimization error, and generalize well to unseen data, i.e., to exhibit small generalization error.

To demonstrate the capability and effectiveness of learning nonlinear operators by neural networks, we set up the problem as general as possible by using the weakest possible constraints on the sensors and training dataset. Specifically, the only condition required is that the sensor locations $\{x_1, x_2, \ldots, x_m\}$ are the same but not necessarily on a lattice for all input functions u, while we do not enforce any constraints on the output locations y (Fig. 6.1B). However, even this constraint can be lifted and it is only used here for computational expediency. We propose a specific new network architecture, the deep operator network (DeepONet), to achieve small total errors. We will demonstrate that unlike fully-connected neural networks (FNNs), DeepONet significantly improves generalization based on a design of two sub-networks, the *branch net* for the input function and the *trunk-net* for the locations to evaluate the output function. The key point is that we discover a new operator G as a NN, which is able to make inferences for quantities of interest given new and unseen data. If we wish to further interpret the type of operator G using the familiar classical calculus, we can project the results of $G(u)(y)$ onto a

dictionary containing first- or higher-order derivatives, gradients, Laplacians, etc., as it is done currently with existing regression techniques [27, 185]. Returning to the aforementioned example put forward by Gulian et al. [76], using the initial conditions as the function $u(x)$ we can train the operator G offline and then we can predict the dynamics for new unseen boundary conditions with error 10^{-4} or less at a fraction of a second.

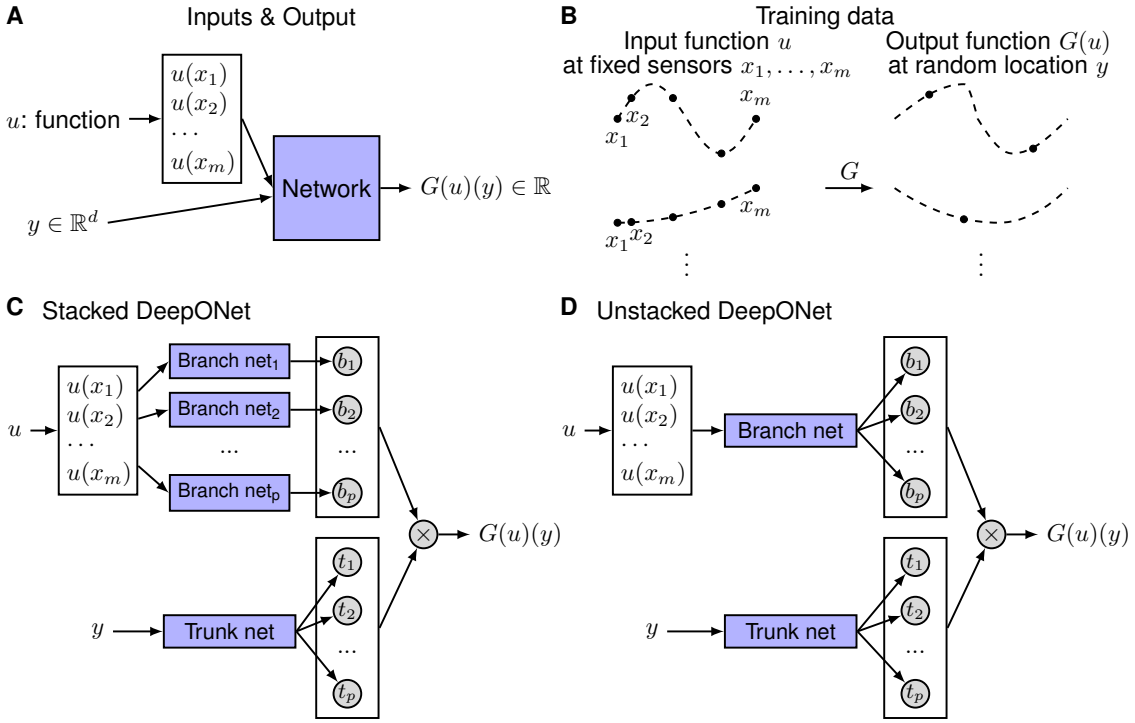


Figure 6.1: A proposal for new architectures that lead to good generalization. Illustrations of the problem setup and architectures of DeepONets. (**A**) For the network to learn an operator $G : u \mapsto G(u)$ it takes two inputs $[u(x_1), u(x_2), \dots, u(x_m)]$ and y. (**B**) Illustration of the training data. For each input function u, we require that we have the same number of evaluations at the same scattered sensors $x_1, x_2, \dots, x_m$. However, we do not enforce any constraints on the number or locations for the evaluation of output functions. (**C**) The stacked DeepONet in Theorem 6.2.1 has one trunk network and p stacked branch networks. (**D**) The unstacked DeepONet has one trunk network and one branch network.

We consider two types of implicit operators, i.e., dynamical systems (e.g., in the form of ordinary differential equations, ODEs) and partial differential equations (PDEs). Some works [172, 240] used standard NNs to identify dynamical systems, but they only considered systems described by difference equations. Some other works [157, 180, 178, 62] predict the evolution of a specific dynamical system

rather than identifying the system behavior for new unseen input signals. The network architectures they employed includes FNNs [180], recurrent neural networks (RNNs) [157], reservoir computing [157], residual networks [178], autoencoder [62], neural ordinary differential equations [38], and neural jump stochastic differential equations [99]. For identifying PDEs, some works treat the input and output function as an image, and then use convolutional neural networks (CNNs) to learn the image-to-image mapping [223, 243], but this approach can be only applied to the particular type of problems, where the sensors of the input function u are distributed on an equispaced grid, and the training data must include all the $G(u)(y)$ values with y also on an equispaced grid. In another approach without this restriction, PDEs are parameterized by unknown coefficients, and then only the coefficient values are identified from data [27, 185, 235, 170, 136]. Alternatively, a generalized CNN based on generalized moving least squares [214] can be used for unstructured data, but it can only approximate local operators and is not able to learn other operators like an integral operator. In the examples we present below we will explore other representations of the discrete function input space.

DeepONet architecture. We focus on learning operators in a more general setting, where the only requirement for the training dataset is the consistency of the sensors $\{x_1, x_2, \dots, x_m\}$ for input functions. In this general setting, the network inputs consist of two separate components: $[u(x_1), u(x_2), \dots, u(x_m)]^T$ and y (Fig. 6.1A), and the goal is to achieve good performance by designing the network architecture. One straightforward solution is to directly employ a classical network, such as FNN, CNN or RNN, and concatenate two inputs together as the network input, i.e., $[u(x_1), u(x_2), \dots, u(x_m), y]^T$. However, the input does not have any specific structure, and thus it is not meaningful to choose networks like CNN or RNN; here we use a FNN as the baseline model.

In high dimensional problems, y is a vector with d components, so the dimension of y does not match the dimension of $u(x_i)$ for $i = 1, 2, \ldots, m$ any more. This also prevents us from treating $u(x_i)$ and y equally, and thus at least two sub-networks are needed to handle $[u(x_1), u(x_2), \ldots, u(x_m)]^T$ and y separately. Although the universal approximation theorem (Theorem 6.2.1) does not have any guarantee on the total error, it still provides us with a network structure in Eq. (6.1). Theorem 6.2.1 only considers a shallow network with one hidden layer, so we extend it to deep networks to gain expressivity. The architecture we propose is shown in Fig. 6.1C. First there is a "trunk" network, which takes y as the input and outputs $[t_1, t_2, \ldots, t_p]^T \in \mathbb{R}^p$. In addition to the trunk network, there are p "branch" networks, and each of them takes $[u(x_1), u(x_2), \ldots, u(x_m)]^T$ as the input and outputs a scalar $b_k \in \mathbb{R}$ for $k = 1, 2, \ldots, p$. We merge them together as in Eq. (6.1): $G(u)(y) \approx \sum_{k=1}^{p} b_k t_k$. We note that the trunk network also applies activation functions in the last layer, i.e., $t_k = \sigma(\cdot)$ for $k = 1, 2, \ldots, p$, and thus this trunk-branch network can also be seen as a trunk network with each weight in the last layer parameterized by another branch network instead of the classical single variable. We also note that in Eq. (6.1) the last layer of each b_k branch network does not have bias. Although bias is not included in Theorem 6.2.1, adding bias may increase the performance by reducing the generalization error (Fig. 6.2). In addition to adding bias to the branch networks, we also add a bias $b_0 \in \mathbb{R}$ in the last stage: $G(u)(y) \approx \sum_{k=1}^{p} b_k t_k + b_0$.

In practice, p is at least of the order of 10, and using lots of branch networks is inefficient. Hence, we merge all the branch networks into one single branch network (Fig. 6.1D), i.e., a single branch network outputs a vector $[b_1, b_2, \ldots, b_p]^T \in \mathbb{R}^p$. In the first DeepONet (Fig. 6.1C), there are p branch networks stacked parallel, so we name it "stacked DeepONet", while we refer to the second DeepONet (Fig. 6.1D)

as "unstacked DeepONet". All versions of DeepONets are implemented in DeepXDE [136], a user-friendly Python library designed for scientific machine learning. The loss function we used is the mean squared error (MSE) between the true value of $G(u)(y)$ and the network prediction for the input $([u(x_1), u(x_2), \dots, u(x_m)], y)$.

DeepONet is a high-level network architecture without defining the architectures of its inner trunk and branch networks. To demonstrate the capability and good performance of DeepONet alone, we choose the simplest FNN as the architectures of the sub-networks in this study. It is possible that using convolutional layers we could further improve accuracy. However, convolutional layers usually work for square domains with $\{x_1, x_2, \dots, x_m\}$ on a equispaced grid, so as alternative and for a more general setting we may use the "attention" mechanism [219].

Embodying some prior knowledge into neural network architectures usually induces good generalization. This inductive bias has been reflected in many networks, such as CNN for images and RNN for sequential data. The success of DeepONet even using FNN as its sub-networks is also due to its strong inductive bias. The output $G(u)(y)$ has two independent inputs u and y, and thus using the trunk and branch networks explicitly is consistent with this prior knowledge. More broadly, $G(u)(y)$ can be viewed as a function of y conditioning on u. Finding an effective way to represent the conditioning input is still an open question, and different approaches have been proposed, such as feature-wise transformations [59].

Data generation. The input function $u(x)$ plays an important role in operator identification; in this study, we mainly consider the following function spaces:

Gaussian random fields (GRF), spectral representations, and formulating the input functions as images; more details are given in Section D.2.

In order to estimate how many sensors are required to achieve accuracy ε, we consider the following ODE system:

$$\textbf{Problem 1} \qquad \begin{cases} \frac{d}{dx}\boldsymbol{s}(x) = \boldsymbol{g}(\boldsymbol{s}(x), u(x), x) \\ \boldsymbol{s}(a) = \boldsymbol{s_0} \end{cases}, \tag{6.2}$$

where $u \in V$ (a compact subset of $C[a,b]$) is the input signal, and $\boldsymbol{s} : [a,b] \to \mathbb{R}^K$ is the solution of system (6.2) serving as the output signal.

Let G be the operator mapping the input u to the output $\boldsymbol{s}$, i.e., $G(u)$ satisfies

$$G(u)(x) = \boldsymbol{s_0} + \int_a^x \boldsymbol{g}(G(u)(t), u(t), t)dt.$$

Now, we choose uniformly $m+1$ points $x_j = a + j(b-a)/m, j = 0, 1, \cdots, m$ from $[a,b]$, and define the function $u_m(x)$ as follows:

$$u_m(x) = u(x_j) + \frac{u(x_{j+1}) - u(x_j)}{x_{j+1} - x_j}(x - x_j),\ x_j \le x \le x_{j+1},\ j = 0, 1, \cdots, m-1.$$

Denote the operator mapping u to u_m by $\mathcal{L}_m$, and let $U_m = \{\mathcal{L}_m(u) | u \in V\}$, which is a compact subset of $C[a,b]$, since V is compact and continuous operator $\mathcal{L}_m$ keeps the compactness. Obviously, $W_m := V \cup U_m$ as the union of two compact sets is also compact. Then, set $W := \bigcup_{i=1}^{\infty} W_i$, and the Lemma in Section D.4 points out that W is still a compact set. Since G is a continuous operator, $G(W)$ is compact in $C([a,b]; \mathbb{R}^K)$. Let $\mathcal{B}[G(W)]$ and $\mathcal{B}[W]$ be the unions of the ranges of the functions in $G(W)$ and W, respectively, and then $\mathcal{B}[G(W)]$ and $\mathcal{B}[W]$ are compact due to the compactness of $G(W)$ and W. For convenience of analysis, we

assume that $\boldsymbol{g}(\boldsymbol{s}, u, x)$ satisfies the Lipschitz condition with respect to $\boldsymbol{s}$ and u on $\mathcal{B}[G(W)] \times \mathcal{B}[W]$, i.e., there is a constant $c > 0$ such that

$$\begin{aligned} \|\boldsymbol{g}(\boldsymbol{s_1}, u, x) - \boldsymbol{g}(\boldsymbol{s_2}, u, x)\|_2 &\leq c\|\boldsymbol{s_1} - \boldsymbol{s_2}\|_2 \\ \|\boldsymbol{g}(\boldsymbol{s}, u_1, x) - \boldsymbol{g}(\boldsymbol{s}, u_2, x)\|_2 &\leq c|u_1 - u_2| \end{aligned}.$$

Note that this condition is easy to achieve, for instance, as long as $\boldsymbol{g}$ is differentiable with respect to $\boldsymbol{s}$ and u on $\mathcal{B}[G(W)] \times \mathcal{B}[W]$.

For $u \in V, u_m \in U_m$, there exists a constant $\kappa(m, V)$ depending on m and compact space V, such that

$$\max_{x \in [a,b]} |u(x) - u_m(x)| \leq \kappa(m, V), \quad \kappa(m, V) \to 0 \ \ as \ \ m \to \infty. \tag{6.3}$$

When V is GRF with the Gaussian kernel, we have $\kappa(m, V) \sim \frac{1}{m^2 l^2}$, see Section D.3 for the proof. Based on the these concepts, we have the following theorem.

Theorem 6.2.2. *Suppose that m is a positive integer making $c(b-a)\kappa(m, V)e^{c(b-a)}$ less than ε, then for any $d \in [a, b]$, there exist $\boldsymbol{\mathcal{W}}_1 \in \mathbb{R}^{n \times (m+1)}, b_1 \in \mathbb{R}^{m+1}, \boldsymbol{\mathcal{W}}_2 \in \mathbb{R}^{K \times n}, b_2 \in \mathbb{R}^K$, such that*

$$\|G(u)(d) - (\boldsymbol{\mathcal{W}}_2 \cdot \sigma(\boldsymbol{\mathcal{W}}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 < \varepsilon$$

holds for all $u \in V$.

Proof. The proof can be found in Section D.4. □

6.3 Results and Discussion

We first show how to learn explicit operators, e.g., integration, fractional derivative and the 2D fractional Laplacian, and demonstrate small generalization error for different representations of the discrete input space V. We then present how to learn implicit operators, including a deterministic PDE and two stochastic differential equations. The parameter values for all examples are listed in Section D.5, and the verification of the Hölder continuity of all the explicit and implicit operators considered in this study is shown in Section D.13.

6.3.1 Learning explicit operators

First, we consider a pedagogical example described by

$$\frac{ds(x)}{dx} = g(s(x), u(x), x), \quad x \in (0, 1],$$

with an initial condition (IC) $s(0) = 0$. Our goal is to predict $s(x)$ over the whole domain $[0, 1]$ for any $u(x)$. We first consider a linear problem by choosing $g(s(x), u(x), x) = u(x)$, which is equivalent to learning the antiderivative operator. That is,

Problem 1.A $$\frac{ds(x)}{dx} = u(x), \quad \text{and} \quad G : u(x) \mapsto s(x) = s_0 + \int_0^x u(\tau) d\tau, \; x \in [0, 1]. \tag{6.4}$$

We train FNNs and residual neural networks (ResNets) [87] to learn the antiderivative operator. To obtain the best performance of FNNs, we grid search the three hyperparameters: depth from 2 to 4, width from 10 to 2560, and learning rate from 0.0001 to 0.01. The mean squared errors (MSE) of the test dataset with learning

rate 0.01, 0.001, and 0.0001 are shown in Fig. D.1. Although we only choose depth up to 4, the results show that increasing the depth further does not improve the test error. Among all these hyperparameters, the smallest test error $\sim 7 \times 10^{-5}$ is obtained for the network with depth 2, width 2560, and learning rate 0.001. We observe that when the network is small, the training error is large and the generalization error (the difference between test error and training error) is small, due to small expressivity. When the network size increases, the training error decreases, but the generalization error increases. We stop the training before FNNs reach the overfitting region, where the test error increases. Similarly, for ResNets, we grid search the two hyperparameters: the number of residual blocks from 1 to 5, and width from 10 to 320. We note that one residual block includes two dense layers with a shortcut connection, and each ResNet has one hidden layer before the first residual bock and one hidden layer after the last residual block [87], and thus a ResNet has in total (3+2# residual block) layers. We choose the learning rate as 0.001 based on the previous results of FNNs. Among all these hyperparameters, the smallest test error $\sim 1 \times 10^{-4}$ is obtained for the ResNet with 1 residual block and width 20.

Compared to FNNs and ResNets, DeepONets have much smaller generalization error and thus smaller test error. Here we do not aim to find the best hyperparameters, and only test the performance of the two DeepONets listed Table D.3. The training trajectory of an unstacked DeepONet with bias is shown in Fig. 6.2B, and the generalization error is negligible. We observe that for both stacked and unstacked DeepONets, adding bias to branch networks reduces both training and test errors (Fig. 6.2A); DeepONets with bias also have smaller uncertainty, i.e., they are more stable for training from random initialization (Fig. 6.2A). Compared to stacked DeepONets, although unstacked DeepONets have

larger training error, the test error is smaller, due to the smaller generalization error. Therefore, unstacked DeepONets with bias achieve the best performance. In addition, unstacked DeepONets have fewer number of parameters than stacked DeepONets, and thus can be trained faster using much less memory. We also provide the examples of Legendre transform in Section D.6 and nonlinear ODE in Section D.7 with more details on the accuracy and architecture. In the following study, we will use unstacked DeepONets.

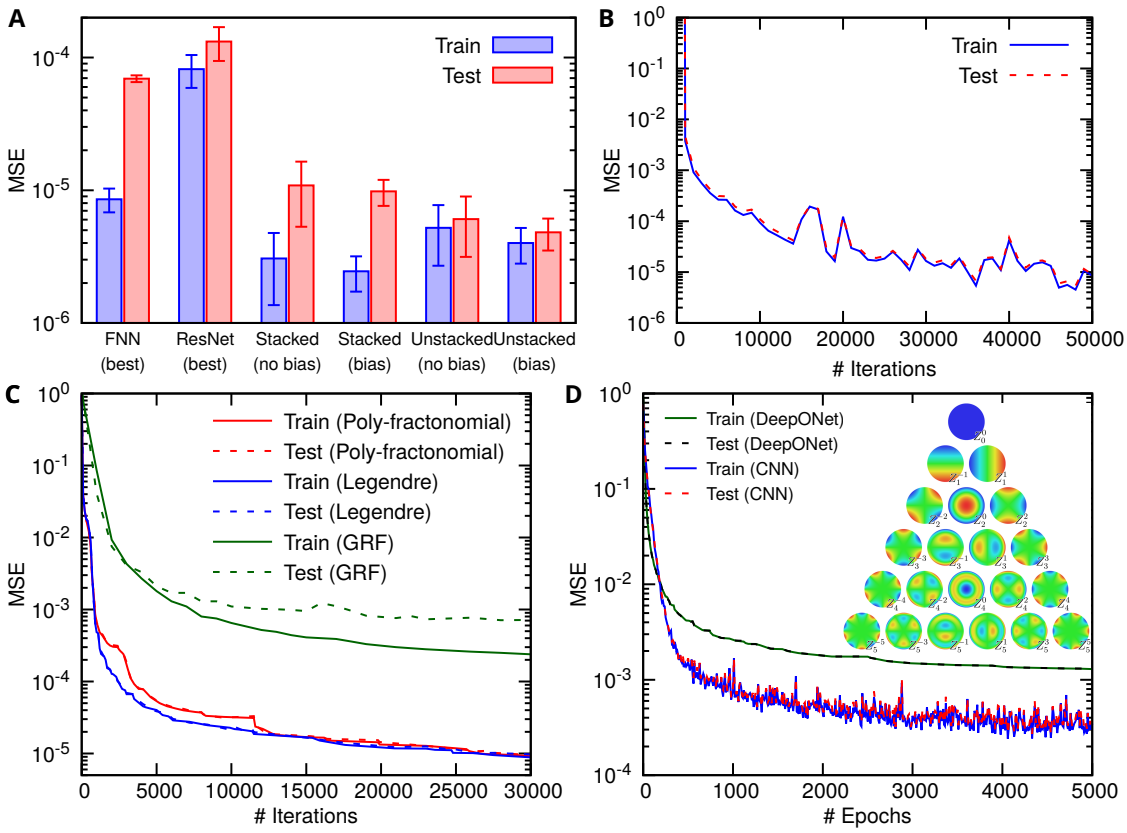


Figure 6.2: Learning explicit Operators using different V spaces and different architectures. Top row: Effect of architectures. Errors of DeepONets trained to learn the antiderivative operator (linear case). (**A**) The training/test errors of stacked/unstacked DeepONets with/without bias compared to the best test error and the corresponding training error of FNNs and ResNets. The error bars are the one-standard-derivation from 10 runs with different training/test data and network initialization. (**B**) The training trajectory of an unstacked DeepONet with bias. Lower row: Effect of space V. (**C**) learning the Caputo fractional derivative: poly-fractonomials vs Legendre vs GRF. (**D**) Learning the fractional Laplacian on a disk. The V space consists of the Zernike polynomials.

In addition to the aforementioned integral operator, we consider integro-differential operators, namely, fractional differential operators, in order to demon-

strate the flexibility of DeepONets to learn more complicated operators. The first fractional differential operator we learn is the 1D Caputo fractional derivative [174]:

Problem 2

$$G(u)(y,\alpha): u(x) \mapsto s(y,\alpha) = \frac{1}{\Gamma(1-\alpha)} \int_0^y (y-\tau)^{-\alpha} u'(\tau) d\tau, \ y \in [0,1], \alpha \in (0,1),$$

where α and $u'(\cdot)$ are the fractional order and first derivative of u, respectively. The domain of output function now includes two variables y and α. We concatenate y and α to form an augmented $\hat{y} = [y, \alpha]^T$ and then feed $\hat{y}$ to the trunk net. We consider the influence of different V spaces on the generalization error of DeepONets. These spaces include two orthogonal polynomial spaces spanned by poly-factonomials [230] and Legendre polynomials, as well as the GRF. More details can be found in Section D.12. Fig. 6.2C shows the generalization errors for the three different V spaces. We see that small generalization errors are achieved for all of the cases. The generalization error for GRF is slightly larger than those for orthogonal polynomial spaces, since for GPR we select different characteristic lengths l's in the RBF kernel for training and test sets.

The second fractional differential operator we learn is the 2D Riesz fractional Laplacian [132]:

Problem 3

$$G(u)(y,\alpha): u(x) \mapsto s(y,\alpha) = \frac{2^\alpha \Gamma(1+\frac{\alpha}{2})}{\pi |\Gamma(-\frac{\alpha}{2})|} \times \text{p.v.} \int_{\mathbb{R}^2} \frac{u(y)-u(\tau)}{\|y-\tau\|_2^{2+\alpha}} d\tau, \quad \alpha \in (0,2),$$

where "p.v." means principle value. The input and out functions are both assumed to be identically zero outside of a unit disk centered at the origin. The 2D fractional

Laplacian reduces to standard Laplacian Δ as the fractional order α goes to two. For learning this operator, we specify the V space to be the orthogonal space spanned by the Zernike polynomials [24], which are commonly used to generate or approximate functions defined on a unit disk. Fig. 6.2D (inset) displays the first 21 Zernike polynomials; see a more detailed description of the polynomial expansions in Section D.2. Importantly, we consider two different NN architectures: trunk and unstacked branch nets versus CNNs. For the first architecture, in a similar manner for handling the 1D Caputo derivative case, we feed the augmented $\hat{y} = [y, \alpha]$ to the trunk net. For the CNN architecture, we rearrange the values of input and output functions to 2D images in which "pixel" (or function) values are attached to a lattice in the polar coordinate. We first utilize a CNN as an encoder to extract the features of the input image, which reduces the high-dimensional input space to a low-dimensional latent space, and then we employ another CNN as a decoder to map the vector in the latent space to the output space. To accommodate the extra parameter α, we set the image consisting of values of $G(y, \alpha_k)$ for k-th α as the k-th channel of the output image. As such, we obtain a multi-channel output image. We observe from Fig. 6.2D that both architectures yield small generalization errors. Moreover, the CNN architecture gains slightly higher operator approximation accuracy than the branch-trunk nets. This is because the former sufficiently takes advantage of the spatial structure of the training data. Nevertheless, DeepONet is more flexible than CNN for unstructed data as we commented in preceding paragraphs. More details are shown in Section D.12.

6.3.2 DeepONet learns fast

An important question for the effectiveness of DeepONet is how fast it learns new operators. We investigate this question by learning a system of ODEs first and subsequently a nonlinear PDE. First, we consider the motion of a gravity pendulum with an external force described by

$$\textbf{Problem 1.B} \qquad \frac{ds_1}{dt} = s_2, \qquad \frac{ds_2}{dt} = -k \sin s_1 + u(t),$$

with an initial condition $\mathbf{s}(0) = \mathbf{0}$, and k determined by the acceleration due to gravity and the length of the pendulum. This problem is characterized by three factors: (1) k, (2) maximum prediction time T, and (3) input function space. The accuracy of learned networks is determined by four factors: (1) the number of sensor points m; (2) training dataset size; (3) network architecture, (4) optimizer. We investigate the effects of all these factors on the accuracy of DeepONet in Section D.8 and verify our analysis on the number of sensors, but here we focus on the convergence rate of the training process.

The test and generalization errors of networks with different width are shown in Fig. 6.3. It is surprising that the test and generalization errors have exponential convergence for small training dataset size. Even for a large dataset, the polynomial convergence rates are still higher than the classical $x^{-0.5}$ in the learning theory [150]. This fast convergence reveals that DeepONets learn exponentially fast, especially in the region of small dataset. Moreover, the transition point from exponential to polynomial convergence depends on the network size, and a larger exponential regime can be accomplished with a sufficiently large network.

Next, we learn an implicit operator in the form of a nonlinear diffusion-reaction

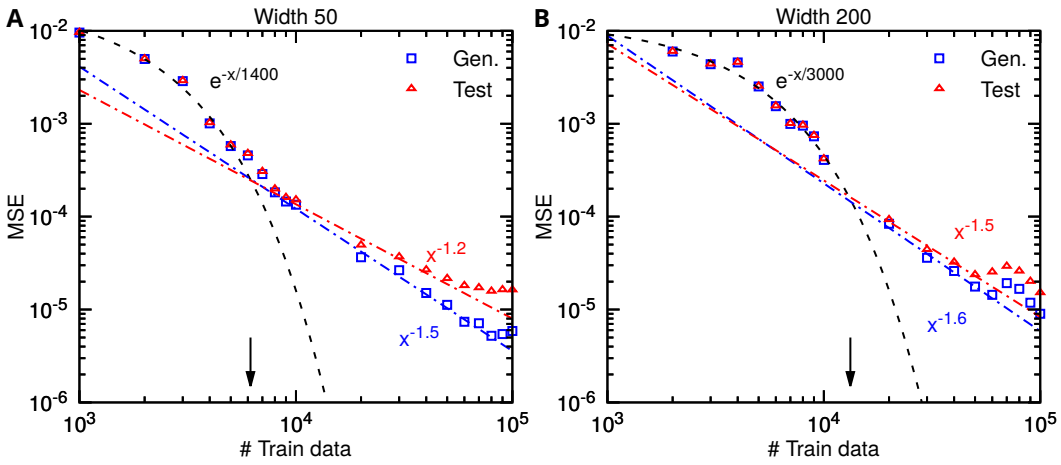


Figure 6.3: Fast learning of implicit operators – a nonlinear pendulum. Initial exponential decay of test/generalization error. The range of exponential convergence increases with the size of the neural network. The test and generalization errors have exponential convergence for small training datasets, and then converge with polynomial rates. The transition point from exponential to polynomial (indicated by the arrow) convergence depends on the width, and a bigger network has a later transition point.

PDE with a source term $u(x)$ described by

$$\textbf{Problem 4} \qquad \frac{\partial s}{\partial t} = D\frac{\partial^2 s}{\partial x^2} + ks^2 + u(x), \quad x \in (0,1), t \in (0,1],$$

with zero initial/boundary conditions, where $D = 0.01$ is the diffusion coefficient, and $k = 0.01$ is the reaction rate. We use DeepONets to learn the operator mapping from $u(x)$ to the PDE solution $s(x,t)$. In the previous examples, for each input u, we only use one random point of $s(x)$ for training, and instead we may also use multiple points of $s(x)$. To generate the training dataset, we solve the diffusion-reaction system using a second-order implicit finite difference method on a 100 by 100 grid, and then for each s we randomly select P points out of these 10000 ($= 100 \times 100$) grid points (Fig. D.8). Hence, the dataset size is equal to the product of P by the number of u samples. We confirm that the training and test datasets do not include the data from the same s.

We investigate the error tendency with respect to the number of u samples and

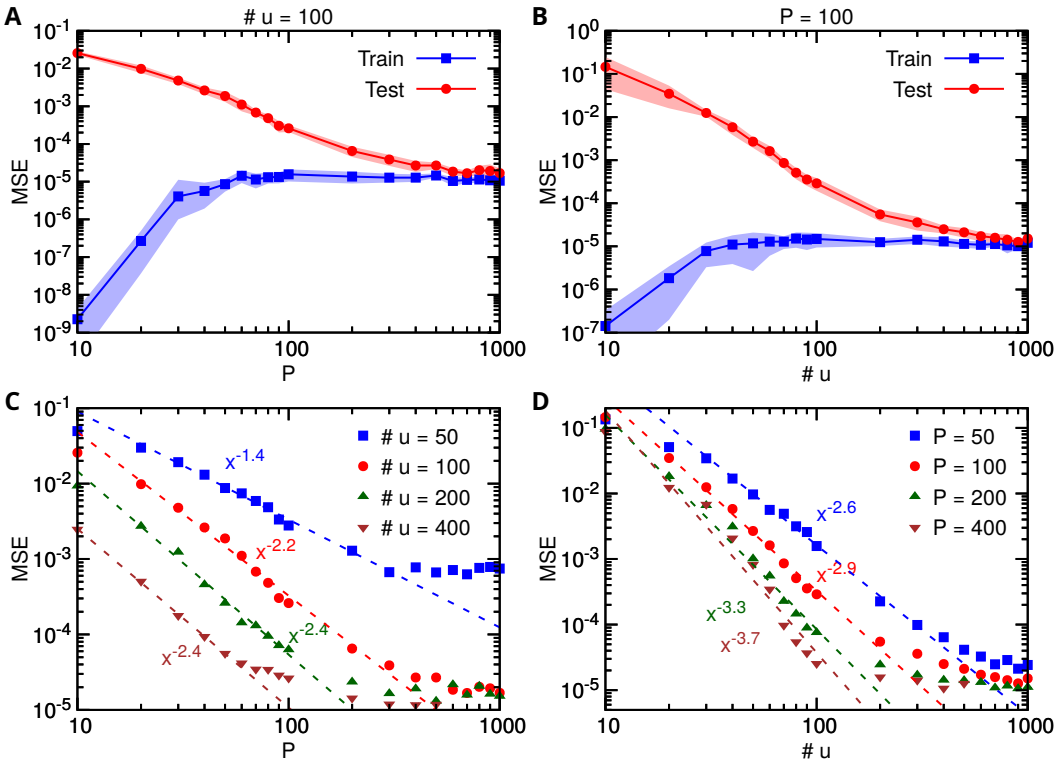


Figure 6.4: Fast Learning of implicit operators – a diffusion-reaction system. Top-row: Comparison of training and testing errors. Learning a diffusion-reaction system. (**A**) Training error (blue) and test error (red) for different values of the number of random points P when 100 random u samples are used. (**B**) Training error (blue) and test error (red) for different number of u samples when $P = 100$. The shaded regions denote one-standard-derivation. Lower row: Learning a reaction-diffusion system. The initial decay is exponential ($P, u < 50$) and the int transitions to algebraic decay in terms of the P sampling point in $x - t$ domain and the excitation function $u(x, t)$. Error convergence rates for different number of training data points. (**C**) Convergence of test error with respect to P for different number of u samples. (**D**) Convergence of test error with respect to the number of u samples for different values of P.

the value of P. When we use 100 random u samples, the test error decreases first as P increases (Fig. 6.4A), and then saturates due to other factors, such as the finite number of u samples and fixed neural network size. We observe a similar error tendency but with less saturation as the number of u samples increases with P fixed (Fig. 6.4B). In addition, in this PDE problem the DeepONet is able to learn from a small dataset, e.g., a DeepONet can reach the test error of $\sim 10^{-5}$ when it is only trained with 100 u samples ($P = 1000$). We recall that we test on 10000 grid points, and thus on average each location point only has $100 \times 1000/10000 = 10$ training data points. For comparison, we train ResNets of different sizes with the same setup, and the test error is of $\sim 10^{-1}$ due to large generalization error (Table D.4).

Before the error saturates, the rates of convergence with respect to both P and the number of u samples obey a polynomial law in most of the range (Figs. 6.4C and D). The rate of convergence versus P depends on the number of u samples, and more u samples induce faster convergence until saturation (see the blue line in Fig. D.9D). Similarly, the rate of convergence versus the number of u samples depends on the value of P (see, the red line in Fig. D.9D). In addition, in the initial range of the convergence, we observe an exponential convergence (see Figs. D.9A and B). The coefficient $1/k$ in the exponential convergence $e^{-x/k}$ also depends on the number of u samples or the value of P (see Fig. D.9C). It is reasonable that the convergence rate presented in Figs. D.9C and D increases with the number of u samples or the value of P, because the total number of training data points is equal to $P \times \#u$. However, by fitting the points, it is surprising that there is a clear tendency in the form of either $\ln(x)$ or e^{-x} (Figs. D.9C and D), which we cannot fully explain yet, and hence more theoretical and computational investigations are required. We also provide the examples of advection equation in Section D.10 and

advection-diffusion equation in Section D.11.

6.3.3 Learning stochastic operators

Next, we demonstrate that we can learn high-dimensional operators, so here we consider a stochastic ODE and a stochastic PDE and present our main findings.

Consider the population growth model

Problem 5 $$dy(t;\omega) = k(t;\omega)y(t;\omega)dt, \quad t \in (0,1] \text{ and } \omega \in \Omega, \tag{6.5}$$

with $y(0) = 1$. Here Ω is the random space. The randomness comes from the coefficient $k(t;\omega)$; here, $k(t;\omega)$ is modeled as a Gaussian random process such that

$$k(t;\omega) \sim \mathcal{GP}(k_0(t), \text{Cov}(t_1, t_2)),$$

where the mean $k_0(t) = 0$ and the covariance function is $\text{Cov}(t_1, t_2) = \sigma^2 \exp(-\|t_1 - t_2\|^2/2l^2)$. We choose $\sigma = 1$, and the correlation length l is in the range $[1, 2]$.

We use DeepONet to learn the operator mapping from $k(t;\omega)$ of different correlation lengths to the solution $y(t;\omega)$. The main differences between this example and the previous examples are that 1) here the input of the branch net is a random process instead of a function, 2) the input of the trunk net contains both physical spaces and random spaces. Specifically, to handle the random process as the input, we employ the Karhunen-Loeve (KL) expansion,

$$k(t;\omega) \approx \sum_{i=1}^{N} \sqrt{\lambda_i} e_i(t) \xi_i(\omega),$$

where N is the number of retained modes, λ_i and $e_i(t)$ are the i-th largest eigenvalue and its associated normalized eigenfunction of the covariance function, respectively, and $\xi_1, \ldots, \xi_N$ are independent standard Gaussian random variables. Then the input of the branch net is the N eigenfunctions scaled by the eigenvalues

$$[\sqrt{\lambda_1}e_1(t), \sqrt{\lambda_2}e_2(t), \ldots, \sqrt{\lambda_N}e_N(t)] \in \mathbb{R}^{N\times m},$$

where $\sqrt{\lambda_i}e_i(t) = \sqrt{\lambda_i}[(e_i(t_1), e_i(t_2)), \ldots, e_i(t_m)] \in \mathbb{R}^m$, and the input of the trunk net is $[t, \xi_1, \xi_2, \ldots, \xi_N] \in \mathbb{R}^{N+1}$.

We choose $N = 5$, which is sufficient to conserve 99.9% stochastic energy. We train a DeepONet with a dataset of 10000 different $k(t;\omega)$ with l randomly sampled in $[1, 2]$, and for each $k(t;\omega)$ we use only one realization. The test MSE is 8.0×10^{-5} $\pm$ 3.4×10^{-5}, and as an example the prediction for 10 different random samples from $k(t;\omega)$ with $l = 1.5$ is in Fig. 6.5.

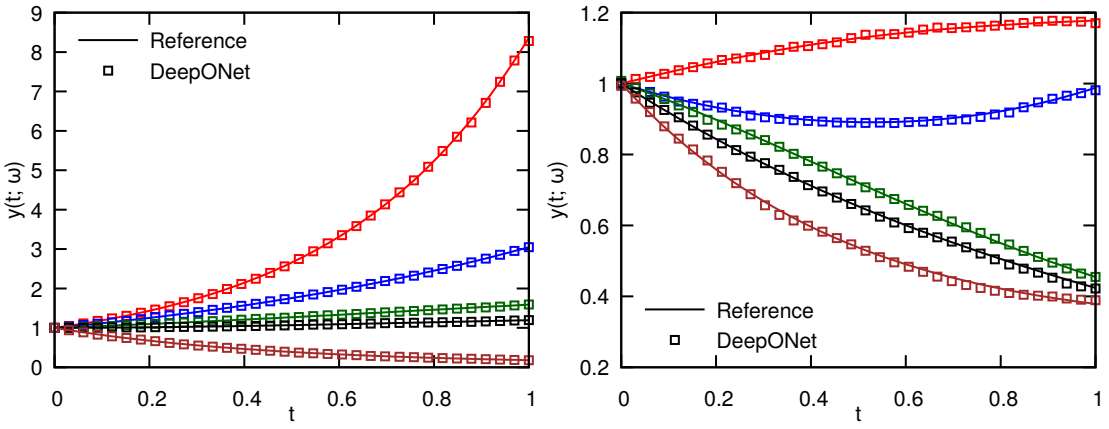


Figure 6.5: DeepONet prediction for a stochastic ODE. The DeepONet prediction (dots) is very close to the reference solution for 10 different random samples from $k(x;\omega)$ with $l = 1.5$.

Next, we consider the following elliptic problem with multiplicative noise

Problem 6 $\quad -\operatorname{div}(e^{b(x;\omega)}\nabla u(x;\omega)) = f(x), \quad x \in (0,1)$ and $\omega \in \Omega$,

with Dirichlet boundary conditions $u(0) = u(1) = 0$. The randomness comes from the diffusion coefficient $e^{b(x;\omega)}$, and $b(x;\omega) \sim \mathcal{GP}(b_0(x), \mathrm{Cov}(x_1, x_2))$, where the mean $b_0(x) = 0$ and the covariance function is $\mathrm{Cov}(x_1, x_2) = \sigma^2 \exp(-\|x_1 - x_2\|^2/2l^2)$. In this example, we choose a constant forcing term $f(x) = 10$, and we set the standard deviation $\sigma = 0.1$ and correlation length in the range $[1, 2]$.

We use DeepONet to learn the operator mapping from $b(x;\omega)$ of different correlation lengths to the solution $u(x;\omega)$. We train a DeepONet with a dataset of 10000 different $b(x;\omega)$ with l randomly sampled in $[1, 2]$, and for each $b(x, \omega)$ we use only one realization. The test MSE is $2.0 \times 10^{-3} \pm 1.7 \times 10^{-3}$, and as an example the prediction for 10 different random samples from $b(x;\omega)$ with $l = 1.5$ is in Fig. 6.6. We have a relative larger error in this stochastic elliptic problem, and to reduce the error further, we can remove outliers of the random variables $\{\xi_1, \xi_2, \ldots, \xi_N\}$ by clipping them to a bounded domain. For instance, if the random variables are clipped to $[-3.1, 3.1]$, which is sufficient large to keep 99% probability space, then the test MSE is reduced to $9.4 \times 10^{-4} \pm 3.0 \times 10^{-4}$.

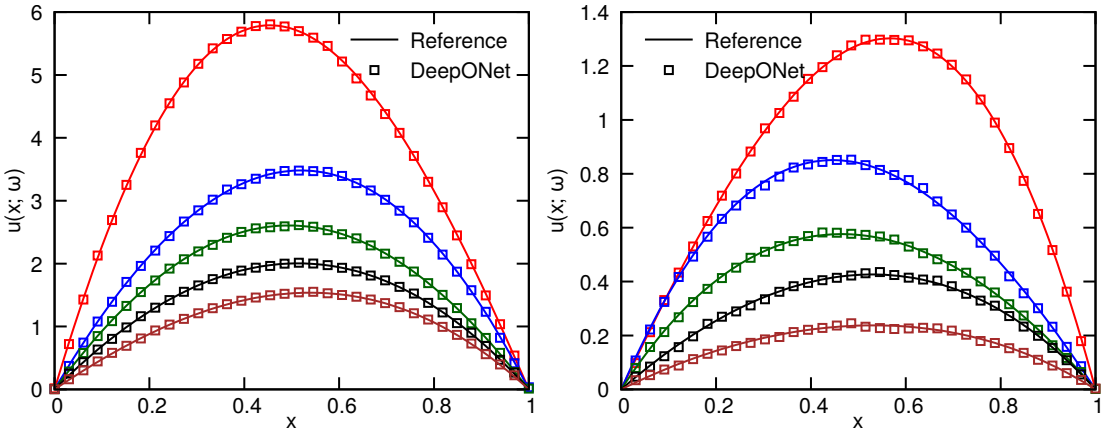


Figure 6.6: DeepONet prediction for a stochastic elliptic equation. The DeepONet prediction (dots) is very close to the reference solution for 10 different random samples from $b(x;\omega)$ with $l = 1.5$.

6.4 Conclusion

We have formulated, for first time, the problem of learning operators in a general setup, and proposed DeepONets to learn linear and nonlinear explicit and implicit operators. In DeepONets, we first construct two sub-networks to encode input functions and location variables separately, and then merge them together to compute the output. We test DeepONets on learning explicit operators, including fractional operators, as well as implicit operators in the form of deterministic and stochastic ordinary/partial differential equations. Our two main findings are that the generalization error is small, and that the training and testing errors decay fast with respect to the training data size. In fact, we observed numerically exponential fast learning for sufficiently large networks and transition to standard convergence for large data sets. It is conceivable that a different type of architecture, a much bigger size or a different type could maintain exponential fast learning for much bigger data sizes. In our simulations, we studied systematically the effects on the test error of different factors, including the number of sensors, maximum prediction time, the complexity of the space of input functions, training dataset size, and network size. Moreover, we derived theoretically the dependence of approximation error on different factors, which is consistent with our computational results.

Despite the reported progress in this chapter, more work should be done both theoretically and computationally. Training DeepOnet is much more computationally intensive than training standard NNs for function approximation, and hence more research is needed in speeding up the training process and in formulating efficient offline training strategies, including perhaps transfer learning approaches. Here, we have employed known operators in order to evaluate the

accuracy of DeepONet systematically, however, the real strength of DeepONet is that it can discover new operators that are trained by multi-fidelity data or by heterogeneous sources of experimental data and simulation data. We could also endow the operator G with prior knowledge, e.g. translational and rotational invariances as in CNN for imaging [211]. On the theoretical side, there have not been any results on the network size for operator approximation, similar to the bounds of width and depth for function approximation [80]. We also do not understand theoretically yet why DeepONets can induce small generalization errors. On the other hand, in this chapter we use fully-connected neural networks for the two sub-networks, but as we discussed in Materials and Methods, we can also employ other network architectures, such as convolutional neural networks or the "attention" mechanism. These modifications may improve further the accuracy of DeepONets, and the example of using CNN with encoders and decoders for learning the fractional Laplacian presented here is a first such indication.

In summary, we have formulated a new framework to learn linear and nonlinear operators implicitly as NNs. In theory, all the operators of the classical integer calculus but also of fractional calculus can be represented by carefully trained NNs as well as other transforms and even theorems, e.g. the Gauss theorem. An even more exciting prospect is to use the big data available in social media to discover new operators that will quantify the currently unknown laws of social dynamics.

Chapter

SEVEN

Conclusion

7.1 Summary

In this dissertation I have presented the theory, algorithms, and open-source software of physics-informed deep learning, as well as the applications to engineering problems.

Dying ReLU. In Chapter 2, we establish, to the best of our knowledge, the first theoretical analysis on the dying ReLU. By focusing on the worst case of dying ReLU, we define 'the dying ReLU network' which refers to the problem when the ReLU network is dead. We categorize the dying process into two phases. One phase is the event where the ReLU network is initialized to be a constant function. We refer to this event as 'the network is born dead'. The other phase is the event where the ReLU network is collapsed after training. Certainly, the first phase implies the second, but not vice versa. We show that the probability that the network is born dead goes to 1 as the depth goes infinite. Also, we provide an upper and a lower bound of the dying probability in $d_{\text{in}} = 1$ when the standard symmetric initialization is used. Furthermore, in order to overcome the dying ReLU networks, we propose a new initialization procedure, namely, a randomized asymmetric initialization (RAI). We show that the RAI has a smaller upper bound of the probability of NNs being born dead. By establishing the expected length map relation (second moment analysis), all parameters needed for the new method are theoretically designed. Numerical examples are provided to demonstrate the performance of our method. We observe that the RAI does not only overcome the dying ReLU but also improves the training and generalization performance.

Generalization. In Chapter 3, we study the generalization error of neural networks for classification problems in terms of the data distribution and neural network smoothness. We first establish a new framework for classification problems. We introduce the *cover complexity* (CC) to measure the difficulty of learning a data set, an accuracy measure called *c-accuracy* which is stronger than the standard classification accuracy, and *the inverse of the modulus of continuity* to quantify neural network smoothness. Subsequently, we derive a quantitative bound for the expected accuracy/error in Theorem 3.3.5, which considers both the cover complexity and neural network smoothness. We validate our theoretical results by several data sets of images. Our numerical results demonstrate that CC is a reliable measure for the difficulty of learning to classify a data set. On the other hand, we observe a clear consistency between test loss and neural network smoothness during the training process. We also show that neural network smoothness decreases when the network depth and width increases, and the effects of depth is more significant than that of width, while the smoothness is insensitive to training dataset size.

Multi-fidelity neural networks. We have demonstrated in Chapter 4 a general framework for extracting elastic and plastic properties of engineering alloys through a suite of unique approaches that combine the latest advances in depth-sensing instrumented indentation with computational simulations of the mechanical properties of materials and the latest developments in deep learning using neural networks. We have shown how different single-fidelity and multi-fidelity approaches can be customized to extract different levels of accuracy, even when only a small set of training data is available. Furthermore, our method establishes how long-recognized and hitherto unaddressed limitations of extracting plastic properties of materials from indentation data, such as uniqueness of the estimated

property values, systematic errors and uncertainties arising from the effects of tip radius of nominally "sharp" indenters, can be overcome to produce a significantly higher level of accuracy and fidelity in the inverse analysis approach. We have introduced in this work three multi-fidelity approaches, along with single, dual and multi-indenter analyses, with the goal of significantly reducing the required number of high-fidelity datasets to achieve a chosen level of accuracy, and to significantly improve the accuracy and reliability of the mechanical properties extracted from depth-sensing instrumented indentation. Specifically, the methods outlined here, are shown to:

1. significantly reduce the number of high-fidelity datasets needed to achieve a chosen level of accuracy;
2. utilize previously established physical and scaling laws to improve the accuracy and training efficiency; and
3. integrate simulation data and experimental data (i.e., data fusion) for training and significantly reducing material and/or experimental setup related systematic errors.

Our results are validated with independent experimental data and sources for different types of wrought aluminum and 3D-printed titanium alloys.

Physics-informed neural networks. In Chapter 5, we present the algorithm, approximation theory, and error analysis of the physics-informed neural networks (PINNs) for solving different types of partial differential equations (PDEs). Compared to the traditional numerical methods, PINNs employ automatic differentiation to handle differential operators, and thus they are mesh-free. Unlike numer-

ical differentiation, automatic differentiation does not differentiate the data and hence it can tolerate noisy data for training. We also discuss how to extend PINNs to solve other types of differential equations, such as integro-differential equations, and also how to solve inverse problems. In addition, we propose a residual-based adaptive refinement (RAR) method to improve the distribution of residual points during the training process, and thus increase the training efficiency.

DeepONet: Learning nonlinear operators. In Chapter 6, we have formulated, for first time, the problem of learning operators in a general setup, and proposed DeepONets to learn linear and nonlinear explicit and implicit operators. In DeepONets, we first construct two sub-networks to encode input functions and location variables separately, and then merge them together to compute the output. We test DeepONets on learning explicit operators, including fractional operators, as well as implicit operators in the form of deterministic and stochastic ordinary/partial differential equations. Our two main findings are that the generalization error is small, and that the training and testing errors decay fast with respect to the training data size. In fact, we observed exponential fast learning for sufficiently large networks and transition to standard convergence for large data sets. It is conceivable that a different type of architecture, a much bigger size or a different type could maintain exponential fast learning for much bigger data sizes. In our simulations, we studied systematically the effects on the test error of different factors, including the number of sensors, maximum prediction time, the complexity of the space of input functions, training dataset size, and network size. Moreover, we derived theoretically the dependence of approximation error on different factors, which is consistent with our computational results.

In summary, we have formulated a new framework to learn linear and nonlinear

operators implicitly as NNs. In theory, all the operators of the classical integer calculus but also of fractional calculus can be represented by carefully trained NNs as well as other transforms and even theorems, e.g. the Gauss theorem. An even more exciting prospect is to use the big data available in social media to discover new operators that will quantify the currently unknown laws of social dynamics.

Open-source software: DeepXDE. To benefit both the education and the computational science communities, we have developed the Python library DeepXDE in Chapter 5, an implementation of PINNs. By introducing the usage of DeepXDE, we show that DeepXDE enables user codes to be compact and follow closely the mathematical formulation. We also demonstrate how to customize DeepXDE to meet new problem requirements. Our numerical examples for forward and inverse problems verify the effectiveness of PINNs and the capability of DeepXDE. Scientific machine learning is emerging as a new and potentially powerful alternative to classical scientific computing, so we hope that libraries such as DeepXDE will accelerate this development and will make it accessible to the classroom but also to other researchers who may find the need to adopt PINN-like methods in their research, which can be very effective especially for inverse problems.

7.2 Future work

The proposed methods shall be further developed and generalized.

Multi-fidelity neural networks. There are several potentially appealing consequences of the results obtained in Chapter 4.

1. The approach described here provides unique pathways to extract critically needed information on mechanical properties, which cannot be easily obtained by other means, in a wide variety of engineering applications involving both structural and functional materials of different types and size scales.

2. With the cost-effectiveness and sophistication of instrumented indentation, robotics and computing tools, the present approaches can readily be incorporated in a wide variety of manufacturing settings (such as those involving 3D printing) for in situ and real-time estimation of material properties.

3. The approach is also highly adaptable and dynamic in that refinements in the choice of a particular deep learning approach can be made "on-the-fly" depending on the processing conditions, specimen geometry, material characteristics, speed of manufacturing, and the level of accuracy sought in the extracted values of properties.

4. The approaches described here can also be further enhanced, with appropriate modifications, to account for such factors as (a) the build-up of residual stresses during the processing of the material, (b) level of anisotropy in material properties, (c) multi-component architectures involving particle-reinforced, fiber-reinforced or layered composite materials, and (d) tailoring of surface and bulk properties through the deliberate introduction of structural, compositional, geometrical and property gradients.

Physics-informed neural networks. Despite the aforementioned advantages in Chapter 5, PINNs still have some limitations. For forward problems, PINNs are currently slower than finite elements but this can be alleviated via offline training [243, 223]. For long time integration, one can also use time-parallel methods

to simultaneously compute on multiple GPUs for shorter time domains. Another limitation is the search for effective neural network architectures, which is currently done empirically by users; however, emerging meta-learning techniques can be used to automate this search, see [244, 64]. Moreover, while here we enforce the strong form of PDEs, which is easy to be implemented by automatic differentiation, alternative weak/variational forms may also be effective, although they require the use of quadrature grids.

DeepONet: Learning nonlinear operators. Despite the reported progress in Chapter 6, more work should be done both theoretically and computationally. Training DeepOnet is much more computationally intensive than training standard NNs for function approximation, and hence more research is needed in speeding up the training process and in formulating efficient offline training strategies, including perhaps transfer learning approaches. Here, we have employed known operators in order to evaluate the accuracy of DeepONet systematically, however, the real strength of DeepONet is that it can discover new operators that are trained by multi-fidelity data or by heterogeneous sources of experimental data and simulation data. We could also endow the operator G with prior knowledge, e.g. translational and rotational invariances as in CNN for imaging [211]. On the theoretical side, there have not been any results on the network size for operator approximation, similar to the bounds of width and depth for function approximation [80]. We also do not understand theoretically yet why DeepONets can induce small generalization errors. On the other hand, in this chapter we use fully-connected neural networks for the two sub-networks, but as we discussed in Materials and Methods, we can also employ other network architectures, such as convolutional neural networks or the "attention" mechanism. These modifications may improve further the accuracy of DeepONets, and the example of using CNN

with encoders and decoders for learning the fractional Laplacian presented here is a first such indication.

Appendix

A

Dying ReLU and Initialization: Theory and Numerical Examples

A.1 Proof of Theorem 2.3.1

The proof starts with the following lemma.

Lemma A.1.1. *Let $\mathcal{N}^L(\boldsymbol{x})$ be a L-layer ReLU neural network with N_ℓ neurons at the ℓ-th layer. Suppose all weights are randomly independently generated from probability distributions satisfying $P(\boldsymbol{W}_j^\ell \boldsymbol{z} = \boldsymbol{0}) = 0$ for any nonzero vector $\boldsymbol{z} \in \mathbb{R}^{N_{\ell-1}}$ and any j-th row of $\boldsymbol{W}^\ell$. Then*

$$P(\mathcal{N}^L(\boldsymbol{x}) \text{ is born dead in } \mathcal{D}) = P(\exists\, \ell \in \{1, \dots, L-1\} \text{ s.t. } \phi(\mathcal{N}^\ell(\boldsymbol{x})) = \boldsymbol{0}\ \forall \boldsymbol{x} \in \mathcal{D}),$$

where $\mathcal{D} \subset B_r(\boldsymbol{0}) = \{\boldsymbol{x} \in \mathbb{R}^{d_{in}} | \|x\| < r\}$ for any $r > 0$.

Proof. Suppose $\mathcal{N}^L(\mathbf{x}) = \mathcal{N}^L(\mathbf{0})$ for all $\mathbf{x} \in \mathcal{D} \subset B_r(\mathbf{0})$. Then $\phi(\mathcal{N}^{L-1}(\mathbf{x})) = \phi(\mathcal{N}^{L-1}(\mathbf{0}))$ for all $\mathbf{x} \in \mathcal{D}$. If $\phi(\mathcal{N}^{L-1}(\mathbf{x})) = \phi(\mathcal{N}^{L-1}(\mathbf{0})) = \mathbf{0}$, we are done as $\ell = L-1$. If it is not the case, there exists j in $\{1, \cdots, N_{L-1}\}$ such that for all $\mathbf{x} \in \mathcal{D}$,

$$\begin{aligned}(\mathcal{N}^{L-1}(\mathbf{x}))_j = \boldsymbol{W}_j^{L-1}\phi(\mathcal{N}^{L-2}(\mathbf{x})) + \boldsymbol{b}_j^{L-1} &= \boldsymbol{W}_j^{L-1}\phi(\mathcal{N}^{L-2}(\mathbf{0})) + \boldsymbol{b}_j^{L-1} \\ &= (\mathcal{N}^{L-1}(\mathbf{0}))_j > 0.\end{aligned}$$

Thus we have $\boldsymbol{W}_j^{L-1}\left(\phi(\mathcal{N}^{L-2}(\mathbf{x})) - \phi(\mathcal{N}^{L-2}(\mathbf{0}))\right) = 0$ for all $\mathbf{x} \in \mathcal{D}$. Let consider the following events:

$$\begin{aligned}G_{L-1} &:= \{\boldsymbol{W}_j^{L-1}\phi(\mathcal{N}^{L-2}(\mathbf{x})) = \boldsymbol{W}_j^{L-1}\phi(\mathcal{N}^{L-2}(\mathbf{0})), \forall \mathbf{x} \in \mathcal{D}\}, \\ R_{L-2} &:= \{\phi(\mathcal{N}^{L-2}(\boldsymbol{x})) = \phi(\mathcal{N}^{L-2}(\mathbf{0})), \forall \mathbf{x} \in \mathcal{D}\}.\end{aligned}$$

Note that $P(G_{L-1}|R_{L-2}) = 1$. Also, since $P(\boldsymbol{W}^\ell \boldsymbol{z}) = 0$ for any nonzero vector $\boldsymbol{z}$, we

have $P(G_{L-1}|R_{L-2}^c) = 0$. Therefore,

$$P(G_{L-1}) = P(G_{L-1}|R_{L-2})P(R_{L-2}) + P(G_{L-1}|R_{L-2}^c)P(R_{L-2}^c) = P(R_{L-2}).$$

Thus we can focus on $\phi(\mathcal{N}^{L-2}(\mathbf{x})) = \phi(\mathcal{N}^{L-2}(\mathbf{0})), \forall \mathbf{x} \in \mathcal{D}$. If $\phi(\mathcal{N}^{L-2}(\mathbf{x})) = \phi(\mathcal{N}^{L-2}(\mathbf{0})) = \mathbf{0}$, we are done as $\ell = L-2$. If it is not the case, it follows from the similar procedure that $\phi(\mathcal{N}^{L-3}(\mathbf{x})) = \phi(\mathcal{N}^{L-3}(\mathbf{0}))$ in $\mathcal{D}$. By repeating these, we conclude that

$$P(\mathcal{N}^{\ell}(\mathbf{x}) \text{ dies in } \mathcal{D}) = P(\exists\, \ell \in \{1, \dots, L-1\} \text{ such that } \phi(\mathcal{N}^{\ell}(\mathbf{x}))) = \mathbf{0}\ \forall \mathbf{x} \in \mathcal{D}).$$

□

Proof. Let $\mathcal{D} \subset B_r(0) \subset \mathbb{R}^{d_{\text{in}}}$ be a training domain where r is any positive real number. We consider a probability space $(\Omega, \mathcal{F}, P)$ where all random weight matrices and bias vectors are defined on. For every $\ell \geq 1$, let $\mathcal{F}_\ell$ be a sub-σ-algebra of $\mathcal{F}$ generated by $\{\boldsymbol{W}^j, \boldsymbol{b}^j\}_{1 \leq j \leq \ell}$. Since $\mathcal{F}_k \subset \mathcal{F}_\ell$ for $k \leq \ell$, $(\mathcal{F}_\ell)$ is a filtration. Let us define the events of our interest $\{A_\ell\}_{2 \leq \ell}$ where

$$\begin{aligned} A_\ell &= \{\mathcal{N}^{\ell}(\mathbf{x}) \text{ is born dead in } \mathcal{D}\} \\ &\overset{a.s.}{=} \{\exists\, j \in \{1, \dots, \ell-1\} \text{ such that } \phi(\mathcal{N}^{j}(\mathbf{x})) = \mathbf{0}\ \forall \mathbf{x} \in \mathcal{D}\} \end{aligned} \tag{A.1}$$

where the second equality is from Lemma A.1.1. Note that A_ℓ is measurable in $\mathcal{F}_{\ell-1}$. Here $\{\boldsymbol{b}^\ell\}_{1 \leq \ell}$ could be either 0 or random vectors. Since $\mathcal{N}^1(\mathbf{x}) = \boldsymbol{W}^1\mathbf{x} + \boldsymbol{b}^1$, $P(A_1) = 0$. To calculate $P(A_\ell)$ for $\ell \geq 2$, let consider another event $C_{\ell,k}$ on which exactly $(N_\ell - k)$-components of $\phi(\mathcal{N}^\ell(\mathbf{x}))$ are zero on $\mathcal{D}$. For notational completeness, we set $C_{1,k} = \emptyset$ for $0 \leq k < N_1$ and $C_{1,N_1} = \Omega$. Then since

$C_{\ell-1,0} \subset A_\ell$, we have

$$P(A_\ell) \geq P(C_{\ell-1,0}). \tag{A.2}$$

We want to show $\lim_{\ell\to\infty} P(C_{\ell,0}) = 1$. Since $\{C_{\ell-1,k}\}_{0\leq k\leq N_{\ell-1}}$ is a partition of Ω, by the total law of probability, we have

$$P(C_{\ell,s}) = \sum_{k=0}^{N_{\ell-1}} P(C_{\ell,s}|C_{\ell-1,k})P(C_{\ell-1,k}),$$

where $P(C_{\ell,0}|C_{\ell-1,0}) = 1$, and $P(C_{\ell,s}|C_{\ell-1,0}) = 0, \forall s \geq 1$. Since $\boldsymbol{W}_{ij}^\ell$ and $\boldsymbol{b}_j^\ell$ are independently initialized, we have

$$P(C_{\ell,0}|C_{\ell-1,k}) = \left(P(\langle \boldsymbol{W}_j^\ell, \phi(\boldsymbol{\mathcal{N}}^{\ell-1}(\mathbf{x}))\rangle + \boldsymbol{b}_j^\ell < 0|C_{\ell-1,k})\right)^{N_\ell} \geq p_k^{N_\ell} > 0,$$

where the second and the third inequalities hold from the assumption. Here $p_k > 0$ does not depend on ℓ. If $\boldsymbol{W}_{ij}^\ell, \boldsymbol{b}_j^\ell$ are randomly initialized from symmetric distributions around 0,

$$P(C_{\ell,0}|C_{\ell-1,k}) \geq \begin{cases} (2^{-k})^{N_\ell} & \text{if } \boldsymbol{b}^\ell = 0 \\ (2^{-(k+1)})^{N_\ell} & \text{if } \boldsymbol{b}^\ell \text{ is generated from a symmetric distribution} \end{cases}$$

Let define a transition matrix V_ℓ of size $(N_{\ell-1}+1)\times(N_\ell+1)$ such that the $(i+1, j+1)$-component is defined to be

$$V_\ell(i+1, j+1) = P(C_{\ell,j}|C_{\ell-1,i}), \qquad \text{where} \quad 0 \leq j \leq N_\ell \quad \text{and} \quad 0 \leq i \leq N_{\ell-1}.$$

Then given

$$\pi_1 = [P(C_{1,0}), P(C_{1,1}), \cdots, P(C_{1,N_1})] = [0, \cdots, 0, 1] \in \mathbb{R}^{N_1+1},$$

we have

$$\pi_\ell = \pi_1 V_2 \cdots V_\ell = [P(C_{\ell,0}), P(C_{\ell,1}), \cdots, P(C_{\ell,N_\ell})], \qquad \ell \geq 2.$$

Suppose $N_\ell = N$ for all $\ell \geq 1$. Note that the first row of V_ℓ is $[1, 0, \cdots, 0]$ for all $\ell \geq 2$. Thus we have the following strictly increasing sequence $\{a_\ell\}_{\ell=1}^{\infty}$:

$$a_\ell := (\pi_\ell)_1 = P(C_{\ell,0}).$$

Since $a_\ell \leq 1$, it converges, say, $\lim_{\ell\to\infty} a_\ell = a \leq 1$. Suppose $a \neq 1$, i.e., $a - 1 < 0$ and let $p_* = \min_{1\leq k\leq N} p_k$. Then since $a_{k+1} = (\pi_k V_{k+1})_1$, we have

$$a_{k+1} = a_k + \sum_{j=1}^{N} P(C_{k+1,0}|C_{k,j})(\pi_k)_{j+1} \geq a_k + (1 - a_k)(p_*^{-(N+1)})^N.$$

Thus

$$0 \leq a - a_{k+1} \leq a - a_k + (a_k - 1)(p_*^{-(N+1)})^N.$$

By taking limit on the both sides, we have

$$0 \leq (a - 1)(p_*^{-(N+1)})^N < 0,$$

which leads a contradiction. Therefore, $a = \lim_{\ell\to\infty} P(C_{\ell,0}) = 1$. It then follows from Equation A.2 that

$$\lim_{\ell\to\infty} P(\mathcal{N}^\ell(\mathbf{x}) \text{ is born dead in } \mathcal{D}) \geq \lim_{\ell\to\infty} P(C_{\ell-1,0}) = 1,$$

which completes the proof. □

A.2 Proof of Theorem 2.3.2

Proof. Based on Lemma A.1.1, let consider

$$\begin{aligned}
A_\ell &= \{\exists\, j \in \{1,\dots,\ell-1\} \text{ such that } \phi(\mathcal{N}^\ell(\mathbf{x}))) = \mathbf{0}\ \forall \mathbf{x} \in \mathcal{D}\},\\
A_\ell^c &= \{\forall 1 \le j < \ell \text{ there exists } \mathbf{x} \in \mathcal{D} \text{ such that } \phi(\mathcal{N}^j(\mathbf{x})) \neq \mathbf{0}\},\\
\tilde{A}_{\ell,\mathbf{x}}^c &= \{\forall 1 \le j < \ell,\ \phi(\mathcal{N}^j(\mathbf{x})) \neq \mathbf{0}\},\\
\tilde{A}_{\ell,\mathbf{x}} &= \{\exists\, j \in \{1,\dots,\ell-1\} \text{ such that } \phi(\mathcal{N}^j(\mathbf{x})) = \mathbf{0}\}.
\end{aligned}$$

Then if $\mathbf{x} \in \mathcal{D}$, $\tilde{A}_{\ell,\mathbf{x}}^c \subset A_\ell^c$. Thus it suffices to compute $P(\tilde{A}_{\ell,\mathbf{x}}^c)$ as

$$P(A_\ell) = 1 - P(A_\ell^c) \le 1 - P(\tilde{A}_{\ell,\mathbf{x}}^c).$$

For $\mathbf{x} \neq \mathbf{0}$, let consider

$$U_{j,\mathbf{x}} = \{\phi(\mathcal{N}^j(\mathbf{x})) = \mathbf{0}\}, \qquad U_{j,\mathbf{x}}^c = \{\phi(\mathcal{N}^j(\mathbf{x})) \neq \mathbf{0}\}.$$

Note that $\bigcup_{1\le j<\ell} U_{j,\mathbf{x}} = \tilde{A}_{\ell,\mathbf{x}}$ and $\tilde{A}_{\ell,\mathbf{x}}^c = \bigcap_{1\le j<\ell} U_{j,\mathbf{x}}^c$. Since $P(\tilde{A}_{j,\mathbf{x}}^c|\tilde{A}_{j-1,\mathbf{x}}) = 0$ for all j, we have

$$P(\tilde{A}_{\ell,\mathbf{x}}^c) = P(\tilde{A}_{\ell,\mathbf{x}}^c|\tilde{A}_{\ell-1,\mathbf{x}}^c)P(\tilde{A}_{\ell-1,\mathbf{x}}^c) = \cdots = P(\tilde{A}_{1,\mathbf{x}}^c)\prod_{j=2}^{\ell} P(\tilde{A}_{j,\mathbf{x}}^c|\tilde{A}_{j-1,\mathbf{x}}^c). \tag{A.3}$$

Note that $P(\tilde{A}_{1,\mathbf{x}}^c) = 1$. Also, note that since the rows of $\boldsymbol{W}^\ell$ and $\boldsymbol{b}_j^\ell$ are independent,

$$P(\tilde{A}_{j,\mathbf{x}}|\tilde{A}_{j-1,\mathbf{x}}^c) = \prod_{s=1}^{N_{j-1}} P\left(\boldsymbol{W}_s^{j-1}\phi(\mathcal{N}^{j-2}(\mathbf{x})) + \boldsymbol{b}_s^{j-1} \le 0|\tilde{A}_{j-1,\mathbf{x}}^c\right). \tag{A.4}$$

Since the weight and biases are randomly drawn from symmetry distribution around 0 and $P\left(\boldsymbol{W}_s^{j-1}\phi(\boldsymbol{\mathcal{N}}^{j-2}(\mathbf{x})) + \boldsymbol{b}_s^{j-1} = 0 | \tilde{A}^c_{j-1,\mathbf{x}}\right) = 0$, we obtain

$$P\left(\boldsymbol{W}_s^{j}\phi(\boldsymbol{\mathcal{N}}^{j-1}(\mathbf{x})) + \boldsymbol{b}_s^{j} \leq 0 | \tilde{A}^c_{j-1,\mathbf{x}}\right) = \frac{1}{2}.$$

Therefore, $P(\tilde{A}_{j,\mathbf{x}} | \tilde{A}^c_{j-1,\mathbf{x}}) = 2^{-N_{j-1}}$ and thus, $P(\tilde{A}^c_{j,\mathbf{x}} | \tilde{A}^c_{j-1,\mathbf{x}}) = 1 - 2^{-N_{j-1}}$. It then follows from Equation A.3 that

$$P(\tilde{A}^c_{\ell,\mathbf{x}}) = \prod_{j=1}^{\ell-1}(1 - 2^{-N_j}),$$

which completes the proof. □

A.3 Proof of Theorem 2.3.3

Proof. We now assume $d_{\text{in}} = 1$, $N_\ell = N$ and without loss of generality let $\mathcal{D} \subset [-r, r]$ be a training domain for any $r > 0$. Also we assume that all weights are initialized from continuous symmetric probability distributions around 0. And the biases are set to zeros.

Since $d_{\text{in}} = 1$, for each ℓ, there exist non-negative vectors $\boldsymbol{v}_\pm \in \mathbb{R}_+^{N_\ell}$ such that

$$\phi(\boldsymbol{\mathcal{N}}^\ell(x)) = \boldsymbol{v}_+\phi(x) + \boldsymbol{v}_-\phi(-x). \tag{A.5}$$

Let $B_{\ell,0}$ be the event where $\phi(\boldsymbol{\mathcal{N}}^\ell(x)) = 0$, and let $B_{\ell,1}$ be the event where

$$\phi(\boldsymbol{\mathcal{N}}^\ell(x)) = \boldsymbol{v}_+\phi(x), \quad \text{or} \quad \boldsymbol{v}_-\phi(-x), \quad \text{or} \quad \boldsymbol{v}(\phi(x) + b\phi(-x))$$

for some $\boldsymbol{v}_\pm, \boldsymbol{v} \in \mathbb{R}_+^{N_\ell}$ and $b > 0$. Let $B_{\ell,2}$ be the event where

$$\phi(\mathcal{N}^\ell(x)) = \boldsymbol{v}_+\phi(x) + \boldsymbol{v}_-\phi(-x)$$

for some linearly independent vectors $\boldsymbol{v}_\pm$. Then it can be checked that $P(B_{\ell+1,k}|B_{\ell,s}) = 0$ for all $2 \geq k > s \geq 0$. Thus, it suffices to consider $P(B_{\ell+1,k}|B_{\ell,s})$ where $0 \leq k \leq s \leq 2$. At $\ell = 1$, since $\phi(\mathcal{N}^1(\mathbf{x})) = \phi(\boldsymbol{W}^1\mathbf{x})$ and $\mathcal{D} \subset [-r, r]$, we have

$$P(B_{1,0}) = 0, \qquad P(B_{1,1}) = 2^{-N+1}, \qquad P(B_{1,2}) = 1 - 2^{-N+1}.$$

For $\ell > 1$, it can be checked that $P(B_{\ell,0}|B_{\ell-1,0}) = 1$, $P(B_{\ell,0}|B_{\ell-1,1}) = 2^{-N}$, and thus $P(B_{\ell,1}|B_{\ell-1,1}) = 1 - 2^{-N}$. For $P(B_{\ell,0}|B_{\ell-1,2})$ and $P(B_{\ell,1}|B_{\ell-1,2})$, we observe the followings. In $B_{\ell,2}$, we have

$$\phi(\langle \boldsymbol{w}, \phi(\mathcal{N}^\ell(x))\rangle) = \phi(\langle \boldsymbol{w}, \boldsymbol{v}_+\rangle)\phi(x) + \phi(\langle \boldsymbol{w}, \boldsymbol{v}_-\rangle)\phi(-x), .$$

Since $\boldsymbol{v}_\pm$ are nonzero vectors in $\mathbb{R}_+^{N_\ell}$ and thus satisfy $\langle \boldsymbol{v}_+, \boldsymbol{v}_-\rangle \geq 0$, by the assumption, we obtain

$$\begin{aligned}
&P(\langle \boldsymbol{w}, \boldsymbol{v}_+\rangle < 0, \langle \boldsymbol{w}, \boldsymbol{v}_-\rangle < 0|\boldsymbol{v}_\pm) = \frac{1}{2} - p_{\boldsymbol{v}_\pm} \geq 1/4,\\
&P(\langle \boldsymbol{w}, \boldsymbol{v}_+\rangle > 0, \langle \boldsymbol{w}, \boldsymbol{v}_-\rangle > 0|\boldsymbol{v}_\pm) = \frac{1}{2} - p_{\boldsymbol{v}_\pm} \geq 1/4,\\
&P(\langle \boldsymbol{w}, \boldsymbol{v}_+\rangle > 0, \langle \boldsymbol{w}, \boldsymbol{v}_-\rangle < 0|\boldsymbol{v}_\pm) = P(\langle \boldsymbol{w}, \boldsymbol{v}_+\rangle < 0, \langle \boldsymbol{w}, \boldsymbol{v}_-\rangle > 0|\boldsymbol{v}_\pm) = p_{\boldsymbol{v}_\pm} \leq \frac{1}{4}.
\end{aligned}$$

Thus we have

$$P(B_{\ell,0}|B_{\ell-1,2}) = \mathbb{E}\left[\left(\frac{1}{2} - p_{\boldsymbol{v}_\pm}\right)^N\right] \geq (1/4)^N$$

where the expectation is taken under $p_{\boldsymbol{v}_\pm}$. For $P(B_{\ell,1}|B_{\ell-1,2})$, there are only three

ways to move from $B_{\ell-1,2}$ to $B_{\ell,1}$. That is

$$\begin{aligned}
B^{(A)}_{\ell,1}|B_{\ell-1,2} &: \phi(\mathcal{N}^{\ell-1}(x)) \to \phi(\mathcal{N}^{\ell}(x)) = \boldsymbol{v}_{+}\phi(x),\\
B^{(B)}_{\ell,1}|B_{\ell-1,2} &: \phi(\mathcal{N}^{\ell-1}(x)) \to \phi(\mathcal{N}^{\ell}(x)) = \boldsymbol{v}_{-}\phi(-x),\\
B^{(C)}_{\ell,1}|B_{\ell-1,2} &: \phi(\mathcal{N}^{\ell-1}(x)) \to \phi(\mathcal{N}^{\ell}(x)) = \boldsymbol{v}(\phi(x) + b\phi(-x)).
\end{aligned}$$

Thus $P(B_{\ell,1}|B_{\ell-1,2}) = P(B^{(A)}_{\ell,1}|B_{\ell-1,2}) + P(B^{(B)}_{\ell,1}|B_{\ell-1,2}) + P(B^{(C)}_{\ell,1}|B_{\ell-1,2})$. Note that due to the symmetry, $P(B^{(A)}_{\ell,1}|B_{\ell-1,2}) = P(B^{(B)}_{\ell,1}|B_{\ell-1,2})$. Thus we have

$$\begin{aligned}
P(B_{\ell,1}|B_{\ell-1,2}) &= 2\left(\sum_{j=1}^{N}\binom{N}{j}\mathbb{E}\left[\left(\frac{1}{2}-p_{\boldsymbol{v}_{\pm}}\right)^{N-j}\left(p_{\boldsymbol{v}_{\pm}}\right)^{j}\right]\right) + \binom{N}{1}\mathbb{E}\left[\left(\frac{1}{2}-p_{\boldsymbol{v}_{\pm}}\right)^{N}\right]\\
&= 2^{-N+1} + (N-2)P(B_{\ell,0}|B_{\ell-1,2})\\
&\geq 2^{-N+1} + (N-2)4^{-N}.
\end{aligned}$$

Note that

$$P(B_{\ell,0}|B_{\ell-1,2}) + P(B_{\ell,1}|B_{\ell-1,2}) = 2^{-N+1} + (N-1)P(B_{\ell,0}|B_{\ell-1,2}).$$

Since $P(A_{\ell+1}) = P(B_{\ell,0})$ where A_ℓ is defined in Equation A.1, we aim to estimate $P(B_{\ell,0})$. Let V_ℓ be the transition matrix whose $(i+1, j+1)$-component is $P(B_{\ell,j}|B_{\ell-1,i})$. Then $P(B_{\ell,0}) = \pi_1 V_2 \cdots V_\ell$ where $(\pi_1)_j = P(B_{1,j-1})$. By letting

$$\gamma_\ell = 1 + \frac{2^{-N} - P(B_{\ell,0}|B_{\ell-1,2})}{2^{-N} + (N-1)P(B_{\ell,0}|B_{\ell-1,2})}.$$

we obtain

$$V_\ell = Q_\ell D_\ell Q_\ell^{-1}, \qquad Q_\ell = \begin{bmatrix} 1/\sqrt{3} & 0 & 0 \\ 1/\sqrt{3} & \frac{1}{\sqrt{1+\gamma_\ell^2}} & 0 \\ 1/\sqrt{3} & \frac{\gamma_\ell}{\sqrt{1+\gamma_\ell^2}} & 1 \end{bmatrix}, \quad Q_\ell^{-1} = \begin{bmatrix} \sqrt{3} & 0 & 0 \\ -\sqrt{1+\gamma_\ell^2} & \sqrt{1+\gamma_\ell^2} & 0 \\ -(1-\gamma_\ell) & -\gamma_\ell & 1 \end{bmatrix}$$

where $D_\ell = \mathrm{diag}(V_\ell)$.

To find a lower bound of $P(B_{\ell,0})$, we consider the following transition matrix $\mathcal{P}$ of size 3×3 which is defined to be

$$\mathcal{P} = \begin{bmatrix} 1 & 0 & 0 \\ (1/2)^N & 1-(1/2)^N & 0 \\ (1/4)^N & (1/2)^{N-1} + (N-2)(1/4)^N & 1-(1/2)^{N-1} - (N-1)(1/4)^N \end{bmatrix}.$$

It can be checked that

$$\mathcal{P} = QDQ^{-1}, \qquad Q = \begin{bmatrix} 1/\sqrt{3} & 0 & 0 \\ 1/\sqrt{3} & \frac{1}{\sqrt{1+\gamma^2}} & 0 \\ 1/\sqrt{3} & \frac{\gamma}{\sqrt{1+\gamma^2}} & 1 \end{bmatrix}, \quad Q^{-1} = \begin{bmatrix} \sqrt{3} & 0 & 0 \\ -\sqrt{1+\gamma^2} & \sqrt{1+\gamma^2} & 0 \\ -(1-\gamma) & -\gamma & 1 \end{bmatrix}$$

where $\gamma = \frac{\mathcal{P}_{32}}{\mathcal{P}_{22}-\mathcal{P}_{33}} = \frac{2^{-N+1}+(N-2)4^{-N}}{2^{-N}+(N-1)4^{-N}} = 1 + \frac{2^{-N}-4^{-N}}{2^{-N}+(N-1)4^{-N}}$ and $D = \mathrm{diag}(\mathcal{P})$. Thus we have

$$\mathcal{P}^\ell = \begin{bmatrix} 1 & 0 & 0 \\ 1-(\mathcal{P}_{22})^\ell & (\mathcal{P}_{22})^\ell & 0 \\ 1-(\mathcal{P}_{22})^\ell - (\gamma-1)((\mathcal{P}_{22})^\ell - (\mathcal{P}_{33})^\ell) & \gamma((\mathcal{P}_{22})^\ell - (\mathcal{P}_{33})^\ell) & (\mathcal{P}_{33})^\ell \end{bmatrix}.$$

Similarly, we obtain

$$V_2 \cdots V_{\ell+1} = \begin{bmatrix} 1 & 0 & 0 \\ 1-(\mathcal{P}_{22})^\ell & (\mathcal{P}_{22})^\ell & 0 \\ \xi_{\ell,31} & \xi_{\ell,32} & \xi_{\ell,33} \end{bmatrix}$$

where $g(x) = 1 - 2^{-N+1} - (N-1)x$, $p_\ell = (V_\ell)_{31} = P(B_{\ell,0}|B_{\ell-1,2})$, $\gamma_{\ell,i} = \gamma_i$ for $1 \le i \le \ell$, $\gamma_{\ell,\ell+1} = 1$,

$$\xi_{\ell,31} = (\mathcal{P}^\ell)_{31} - (\gamma_{\ell,1}-1)(g(p_1))^\ell + \sum_{i=1}^{\ell}(\gamma_{\ell,i} - \gamma_{\ell,i+1})(\mathcal{P}_{22})^{\ell-i}\prod_{j=1}^{i} g(p_j),$$

$$\xi_{\ell,33} = \prod_{j=1}^{\ell} g(p_j), \qquad \xi_{\ell,32} = 1 - \xi_{\ell,31} - \xi_{\ell,33}.$$

We want to show that

$$(\pi_1 \mathcal{P}^\ell)_1 \le (\pi_1 V_2 \cdots V_{\ell+1})_1 = P(B_{\ell+1,0})$$

where

$$\pi_1 = [P(B_{1,0}), P(B_{1,1}), P(B_{1,2})] = [0, 2^{-N+1}, 1-2^{-N+1}].$$

Let denote $\gamma\bar{}_{\ell,i} := \gamma_{\ell,i} - 1$ and $g_i := g(p_i)$. It then suffices to show that

$$\mathcal{J} := \sum_{i=1}^{\ell}(\gamma\bar{}_{\ell,i} - \gamma\bar{}_{\ell,i+1})(\mathcal{P}_{22})^{\ell-i}\prod_{j=1}^{i} g_j - \gamma\bar{}_{\ell,1}(g_1)^\ell \ge 0.$$

Note that $4^{-N} = p_1 \le p_j < 2^{-N}$, $\mathcal{P}_{22} > g(p_j)$, and thus $\mathcal{P}_{22}^\ell > (g(p_1))^\ell \ge \prod_{j=1}^{\ell} g(p_j)$.

Also,

$$\mathcal{P}_{22} - g(p_i) = 2^{-N} + (N-1)p_i, \qquad \bar{\gamma_{\ell,i}} = \frac{2^{-N} - p_i}{2^{-N} + (N-1)p_i} = \frac{2^{-N} - p_i}{\mathcal{P}_{22} - g(p_i)}.$$

Thus we have

$$\begin{aligned}
\mathcal{J} &= \bar{\gamma_{\ell,1}}(g_1\mathcal{P}_{22}^{\ell-1} - g_1^{\ell}) - \sum_{i=2}^{\ell} \bar{\gamma_{\ell,i}}(\mathcal{P}_{22} - g_i)\mathcal{P}_{22}^{\ell-i}\prod_{j=1}^{i-1} g_j \\
&\geq \bar{\gamma_{\ell,1}}(\mathcal{P}_{22}^{\ell} - g_1^{\ell}) - \sum_{i=1}^{\ell} \bar{\gamma_{\ell,i}}(\mathcal{P}_{22} - g_i)\mathcal{P}_{22}^{\ell-i} g_1^{i-1} \\
&\geq \bar{\gamma_{\ell,1}}(\mathcal{P}_{22}^{\ell} - g_1^{\ell}) - \bar{\gamma_{\ell,1}}(\mathcal{P}_{22} - g_1)\frac{\mathcal{P}_{22}^{\ell} - g_1^{\ell}}{\mathcal{P}_{22} - g_1} = 0.
\end{aligned}$$

Therefore,

$$(\mathcal{P}^{\ell})_{31} \leq (V_2 \cdots V_{\ell+1})_{31},$$

which implies that

$$(\pi_1\mathcal{P}^{\ell})_1 \leq (\pi_1 V_2 \cdots V_{\ell+1})_1 = P(B_{\ell+1,0}) = P(A_{\ell+2}).$$

Furthermore, it can be checked that

$$(\pi\mathcal{P}^{\ell})_1 = 1 - (\mathcal{P}_{22})^{\ell} - \left(\frac{(1-2^{-N})(1-2^{-N+1})}{1+(N-1)2^{-N}}\right)((\mathcal{P}_{22})^{\ell} - (\mathcal{P}_{33})^{\ell}).$$

□

A.4 Proof of Theorem 2.3.4

Proof. Since $\mathcal{N}^L(\mathbf{x})$ is a constant function, it follows from Lemma A.1.1 that with probability 1, there exists ℓ such that $\phi(\mathcal{N}^\ell(\mathbf{x})) \equiv \mathbf{0}$. Then the gradients of the loss function with respect to the weights and biases in the $1, \cdots, \ell$-th layers vanish. Hence, the weights and biases in layers $1, \ldots, \ell$ will not change when a gradient based optimizer is employed. This implies that $\mathcal{N}^L(\mathbf{x})$ always remains to be a constant function as $\phi(\mathcal{N}^\ell(\mathbf{x})) \equiv \mathbf{0}$. Furthermore, the gradient method changes only the weights and biases in layers $\ell + 1, \ldots, L$. Therefore, the ReLU NN can only be optimized to a constant function, which minimizes the loss function $\mathcal{L}$. □

A.5 Proof of Theorem 2.4.1

Lemma A.5.1. *Let $\boldsymbol{v} \in \mathbb{R}^{n+1}$ be a vector such that $\boldsymbol{v}_k \sim P$ and $\boldsymbol{v}_{-k} \sim N(0, \sigma^2 \boldsymbol{I}_n)$ where $\boldsymbol{v}_{-k}$ is defined in Equation 2.5. For any nonzero vector $\boldsymbol{x} \in \mathbb{R}^{n+1}$ whose k-th element is positive (i.e., $\boldsymbol{x}_k > 0$), let*

$$\|\tilde{\boldsymbol{x}}_{-k}\|^2 = \frac{\sum_{j \neq k} \boldsymbol{x}_j^2}{\boldsymbol{x}_k^2}, \qquad \tilde{\sigma}^2 = \|\tilde{\boldsymbol{x}}_{-k}\|^2 \sigma^2.$$

Then

$$P\left(\langle \boldsymbol{v}, \boldsymbol{x} \rangle > 0\right) = \begin{cases} \frac{1}{2} + \int_0^M (1 - F_P(t)) \frac{1}{\sqrt{2\pi}\tilde{\sigma}} e^{-\frac{t^2}{2\tilde{\sigma}^2}} dt & \text{if } \tilde{\sigma}^2 > 0 \text{ and } \boldsymbol{x}_k > 0 \\ 1/2 & \text{if } \|\boldsymbol{x}_{-k}\|^2 > 0 \text{ and } \boldsymbol{x}_k = 0 \\ 1 & \text{if } \tilde{\sigma}^2 = 0 \text{ and } \boldsymbol{x}_k > 0, \end{cases} \tag{A.6}$$

where $F_P(t)$ is the cdf of P.

Proof. We first recall some properties of the normal distribution. Let $Y_1, \cdots, Y_n$ be i.i.d. random variables from $N(0, \sigma^2)$ and let $\boldsymbol{Y}_n = (Y_1, \cdots, Y_n)$. Then for any vector $\boldsymbol{a} \in \mathbb{R}^n$,

$$\langle \boldsymbol{Y}_n, \boldsymbol{a} \rangle = \sum_{i=1}^{n} \boldsymbol{a}_i Y_i \sim N(0, \|\boldsymbol{a}\|^2 \sigma^2).$$

Suppose $\boldsymbol{v}$ is a random vector generated in the way described in Subsection 2.4.1. Then for any $\boldsymbol{x} \in \mathbb{R}^{n+1}$,

$$\langle \boldsymbol{v}, \boldsymbol{x} \rangle = \sum_{i \neq k} \boldsymbol{v}_i \boldsymbol{x}_i + \boldsymbol{v}_k \boldsymbol{x}_k = \boldsymbol{x}_k \left(Z + \boldsymbol{v}_k \right),$$

where $\tilde{\sigma}^2 = \|\tilde{\boldsymbol{x}}_{-k}\|^2 \sigma^2$ and $Z \sim N(0, \tilde{\sigma}^2)$. If $\boldsymbol{x}_k > 0$ and $\|\mathbf{x}_{-k}\|^2 > 0$, we have

$$\langle \boldsymbol{v}, \boldsymbol{x} \rangle > 0 \quad \Longleftrightarrow \quad \boldsymbol{v}_k > -Z \overset{d}{=} Z.$$

Therefore, it suffices to compute $P(\boldsymbol{v}_k > Z)$. Let $f_Z(z)$ be the pdf of Z and $f_{\mathrm{P}}(x)$ be the pdf of $\boldsymbol{v}_k \sim \mathrm{P}$. Then

$$\begin{aligned} P(\boldsymbol{v}_k > Z) &= \int_{-\infty}^{\infty} \int_{z}^{M} f_{\mathrm{P}}(x) dx f_Z(z) dz \\ &= \int_{-\infty}^{0} \int_{0}^{M} f_{\mathrm{P}}(x) dx f_Z(z) dz + \int_{0}^{M} \int_{z}^{M} f_{\mathrm{P}}(x) dx f_Z(z) dz \\ &= \frac{1}{2} + \int_{0}^{M} (1 - F_{\mathrm{P}}(z)) f_Z(z) dz, \end{aligned}$$

which completes the proof. □

Proof. For each ℓ and s, let k_s^ℓ be randomly uniformly chosen in $\{1, \cdots, N_{\ell-1} + 1\}$. Let $\boldsymbol{V}_s^\ell = [\boldsymbol{W}_s^\ell, \boldsymbol{b}_s^\ell]$ and $\mathbf{n}^\ell(\mathbf{x}) = \left[\phi(\mathcal{N}^\ell(\mathbf{x})), 1\right]$. Recall that for a vector $\mathbf{v} \in \mathbb{R}^{N+1}$,

$$\mathbf{v}_{-k} := [v_1, \cdots, v_{k-1}, v_{k+1}, \cdots, v_{N+1}].$$

To emphasize the dependency of k_s^ℓ, we denote $\boldsymbol{V}_s^\ell$ whose k-th component is generated from P by $\boldsymbol{V}_s^\ell(k)$.

The proof can be started from Equation A.4. It suffices to compute

$$P(\tilde{A}_{j,\mathbf{x}}|\tilde{A}^c_{j-1,\mathbf{x}}) = \prod_{s=1}^{N_{j-1}} P\left(\boldsymbol{W}_s^{j-1}\phi(\mathcal{N}^{j-2}(\mathbf{x})) + \boldsymbol{b}_s^{j-1} \leq 0|\tilde{A}^c_{j-1,\mathbf{x}}\right).$$

Note that for each s,

$$P\left(\langle \boldsymbol{V}_s^j, \mathbf{n}^{j-1}\rangle \leq 0|\tilde{A}^c_{j-1,\mathbf{x}}\right) = \frac{1}{N_{j-1}+1}\sum_{k=1}^{N_{j-1}+1} P\left(\langle \boldsymbol{V}_s^j(k), \mathbf{n}^{j-1}\rangle \leq 0|\tilde{A}^c_{j-1,\mathbf{x}}\right).$$

Also, from Lemma A.5.1, we have

$$P\left(\langle \boldsymbol{V}_s^j(k), \mathbf{n}^{j-1}\rangle \leq 0|\tilde{A}^c_{j-1,\mathbf{x}}\right) = \frac{1}{2} - \int_0^M (1 - F_P(z))\frac{1}{\sqrt{2\pi}\tilde{\sigma}_{\ell,k}} e^{-\frac{z^2}{2\tilde{\sigma}^2_{\ell,k}}} dz$$

where

$$\tilde{\sigma}^2_{\ell,k} = \frac{\|\mathbf{n}_{-k}^{\ell-1}(\mathbf{x}))\|^2\sigma_\ell^2}{\left(\mathbf{n}_k^{\ell-1}(\mathbf{x})\right)^2}.$$

Note that if $\mathbf{n}_k^{\ell-1}(\mathbf{x}) = 0$, the above probability is simply $1/2$. Also, since the event $\tilde{A}^c_{j-1,\mathbf{x}}$ is given, we know that $\phi(\mathcal{N}^\ell(\mathbf{x})) \neq 0$. Thus,

$$P\left(\langle \boldsymbol{V}_s^j, \mathbf{n}^{j-1}\rangle \leq 0|\tilde{A}^c_{j-1,\mathbf{x}}\right) < \frac{1}{2}.$$

We thus denote

$$\left(\frac{1}{2} - \delta_{\ell-1,\mathbf{x}}(\omega)\right)^{N_{\ell-1}} := P(\tilde{A}_{\ell,\mathbf{x}}|\tilde{A}^c_{\ell-1,\mathbf{x}}(\omega)),$$

where $\delta_{\ell-1,\mathbf{x}}$ is $\mathcal{F}_{\ell-2}$ measurable whose value lies in $(0, 0.5]$ and it depends on $\mathbf{x}$. It

then follows from Equation A.3 that

$$P(\tilde{A}^c_{\ell,\mathbf{x}}) = \int \prod_{j=1}^{\ell-1} \left(1 - \left(\frac{1}{2} - \delta_{j,\mathbf{x}}(\omega)\right)^{N_j}\right) dP(\omega)$$

where P is the probability distribution with respect to $\{\boldsymbol{W}^j, \boldsymbol{b}^j\}_{1\le j\le \ell}$. Note that since the weight matrix and the bias vector of the first layer is initialized from a symmetric distribution, $\delta_{1,\mathbf{x}} = 0$. Let $\delta^*_{1,\mathbf{x}} = 0$. By the mean value theorem, there exists some numbers $\delta^*_{2,\mathbf{x}}, \cdots, \delta^*_{\ell-1,\mathbf{x}} \in (0, 0.5]$ such that

$$P(\tilde{A}^c_{\ell,\mathbf{x}}) = \prod_{j=1}^{\ell-1} \left(1 - \left(\frac{1}{2} - \delta^*_{j,\mathbf{x}}\right)^{N_{j-1}}\right).$$

Let $\boldsymbol{\delta}^*_{\mathbf{x}} = (\delta^*_{1,\mathbf{x}}, \cdots, \delta^*_{\ell-1,\mathbf{x}})$. By setting

$$\boldsymbol{\delta}^* = \boldsymbol{\delta}^*_{\mathbf{x}^*}, \qquad \text{where} \quad \mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathcal{D}\setminus\{\mathbf{0}\}} \prod_{j=1}^{\ell-1} \left(1 - \left(\frac{1}{2} - \delta^*_{j,\mathbf{x}}\right)^{N_{j-1}}\right),$$

the proof is completed. □

A.6 Proof of Theorem 2.4.3

Let $X_\ell \sim \mathrm{P}_\ell$ whose pdf is denoted by $f_{P_\ell}(x)$. Suppose $0 < \mu'_{\ell,i} = E[X^i_\ell] < \infty$ for $i = 1, 2$. We then define three probability distribution functions:

$$f^{(2)}_{P_\ell}(x) = \frac{x^2}{\mu'_{\ell,2}} f_{P_\ell}(x), \qquad f^{(1)}_{P_\ell}(x) = \frac{x}{\mu'_{\ell,1}} f_{P_\ell}(x), \qquad f^{(0)}_{P_\ell}(x) = f_{P_\ell}(x).$$

Also we denote its corresponding cdfs by $F^{(i)}_{P_\ell}(t) = \int_0^t f^{(i)}_{P_\ell}(x)dx$. For notational convenience, let us assume that $\mathrm{P}_\ell = \mathrm{P}$ and $M_\ell = M$ for all ℓ and define

$$\mathbf{n}^\ell(\mathbf{x}) := \left[\phi(\mathcal{N}^\ell(\mathbf{x})), 1\right] \in \mathbb{R}_+^{N_\ell+1}.$$

We denote the pdf of normal distributions by $f_Y^{\ell,k}(y)$ where

$$f_Y^{\ell,k}(y) = \frac{1}{\sqrt{2\pi}\tilde{\sigma}_{\ell,k}} e^{-\frac{y^2}{2\tilde{\sigma}^2_{\ell,k}}}, \qquad \tilde{\sigma}^2_{\ell,k} = \frac{\|\mathbf{n}^{\ell-1}_{-k}(\mathbf{x}))\|^2\sigma^2_\ell}{\left(\mathbf{n}^{\ell-1}_k(\mathbf{x})\right)^2}, \qquad \sigma^2_\ell = \frac{\sigma^2_w}{N_{\ell-1}}$$

where $1 \le k \le N_{\ell-1}+1$. Recall that for a vector $\mathbf{v} \in \mathbb{R}^{N+1}$,

$$\mathbf{v}_{-k} := [v_1, \cdots, v_{k-1}, v_{k+1}, \cdots, v_{N+1}].$$

For any probability distribution P defined on $[0, M]$ whose pdf is denoted by $f_P(x)$, let

$$\mathcal{J}_{\mathrm{P}}^{\ell,k} := \int_0^M \int_y^M f_P(x)dx f_Y^{\ell,k}(y)dy. \tag{A.7}$$

Let $Y_{\ell,k} \sim N(0, \tilde{\sigma}^2_{\ell,k})$ and $X \sim \mathrm{P}$. Then

$$\begin{aligned} P\left(X > Y_{\ell,k} | \tilde{\sigma}^2_{\ell,k}\right) &= \int_{-\infty}^{\infty}\int_0^M 1_{\{X>Y_{\ell,k}\}} f_P(x) f_Y^{\ell,k}(y)dxdy \\ &= \int_{-\infty}^{0}\int_0^M f_P(x)dx f_Y^{\ell,k}(y)dy + \int_0^M\int_y^M f_P(x)dx f_Y^{\ell,k}(y)dy \\ &= \frac{1}{2} + \mathcal{J}_{\mathrm{P}}^{\ell,k}. \end{aligned}$$

Therefore, $0 \le \mathcal{J}_{\mathrm{P}}^{\ell,k} \le 0.5$.

The proof of Theorem 2.4.3 starts with the following lemma.

Lemma A.6.1. *Suppose* $Y \sim N(0, \tilde{\sigma}^2_{\ell,k})$, $X \sim P_\ell$ *and* $\mu'_{\ell,1} \geq M/2$. *Then*

$$\int_0^M \int_y^M (x-y)^2 f_{P_\ell}(x)dx f_Y^{\ell,k}(y)dy \leq \mu'_{\ell,2}/2.$$

Proof. Let $I_{\ell,k}$ be the quantity of our interest:

$$I_{\ell,k} := \int_0^M \int_y^M (x-y)^2 f_{P_\ell}(x)dx f_Y^{\ell,k}(y)dy.$$

Without loss of generality, we can assume $M = 1$. This is because

$$\begin{aligned}
I_{\ell,k} &= \int_0^M \int_y^M (x-y)^2 f_{P_\ell}(x) f_Y^{\ell,k}(y)dxdy \\
&= \int_0^M \int_{y/M}^1 (Mu-y)^2 f_{P_\ell}(Mu) f_Y^{\ell,k}(y)Mdudy \\
&= \int_0^1 \int_s^1 M^2(u-s)^2 (Mf_{P_\ell}(Mu))(Mf_Y^{\ell,k}(Ms))duds \\
&= M^2 \int_0^1 \int_s^1 (u-s)^2 f_U(u) f_S^{\ell,k}(s)duds \\
&= M^2 \tilde{I}_{\ell,k}
\end{aligned}$$

where $u = x/M$ and $s = y/M$. Here $MU \sim \mathrm{P}_\ell$ and $S \sim N(0, \tilde{\sigma}^2_{\ell,k}/M^2)$.

Let us first consider the inner integral. Let $G(y) = \int_y^1 (x^2 - 2xy + y^2) f_{P_\ell}(x)dx$.

It then can be written as follow:

$$
\begin{aligned}
G(y) &= \int_y^1 (x^2 - 2xy + y^2) f_{P_\ell}(x) dx = \int_y^1 (x^2 f_{P_\ell}(x) - 2yx f_{P_\ell}(x) + y^2 f_{P_\ell}(x)) dx \\
&= \int_y^1 \left(\mu'_{\ell,2} f^{(2)}_{P_\ell}(x) - 2\mu'_{\ell,1} y f^{(1)}_{P_\ell}(x) + y^2 f^{(0)}_{P_\ell}(x) \right) dx \\
&= \mu'_{\ell,2}(1 - F^{(2)}_{P_\ell}(y)) - 2y\mu'_{\ell,1}(1 - F^{(1)}_{P_\ell}(y)) + y^2(1 - F^{(0)}_{P_\ell}(y)) \\
&= -2\mu'_{\ell,1} y + y^2 + \mu'_{\ell,2} \int_y^1 f^{(2)}_{P_\ell}(x) dx + 2\mu'_{\ell,1} y F^{(1)}_{P_\ell}(y) - y^2 F^{(0)}_{P_\ell}(y).
\end{aligned}
$$

Note that since $0 \le y \le 1$, and thus $y^2 \le y$, we have

$$
\begin{aligned}
G(y) &= y^2(1 - F^{(0)}_{P_\ell}(y)) - 2\mu'_{\ell,1} y(1 - F^{(1)}_{P_\ell}(y)) + \mu'_{\ell,2} \int_y^1 f^{(2)}_{P_\ell}(x) dx \\
&\le y(1 - F^{(0)}_{P_\ell}(y)) - 2\mu'_{\ell,1} y(1 - F^{(1)}_{P_\ell}(y)) + \mu'_{\ell,2} \int_y^1 f^{(2)}_{P_\ell}(x) dx \\
&= -y(2\mu'_{\ell,1} - 1) \left[1 - \left(\frac{2\mu'_{\ell,1} F^{(1)}_{P_\ell}(y) - F^{(0)}_{P_\ell}(y)}{2\mu'_{\ell,1} - 1} \right) \right] + \mu'_{\ell,2} \int_y^1 f^{(2)}_{P_\ell}(x) dx.
\end{aligned}
$$

Since $2\mu'_{\ell,1} \ge 1$ and

$$
1 - \left(\frac{2\mu'_{\ell,1} F^{(1)}_{P_\ell}(y) - F^{(0)}_{P_\ell}(y)}{2\mu'_{\ell,1} - 1} \right) \ge 0, \qquad \forall y \in [0,1],
$$

we have $G(y) \le \mu'_{\ell,2} \int_y^1 f^{(2)}_{P_\ell}(x) dx$. Therefore,

$$
I_{\ell,k} \le \mu'_{\ell,2} \int_0^1 \int_y^1 f^{(2)}_{P_\ell}(x) dx f^{\ell,k}_Y(y) dy = \mu'_{\ell,2} \mathcal{J}^{\ell,k}_{f^{(2)}_{P_\ell}}
$$

where $\mathcal{J}^{\ell,k}_{\mathrm{P}}$ is defined in Equation A.7. Since $0 \le \mathcal{J}^{\ell,k}_{\mathrm{P}} \le 0.5$, we conlcude that $I_{\ell,k} \le \mu'_{\ell,2}/2$. □

Proof. It follows from

$$\mathcal{N}_j^\ell(\mathbf{x}) = \boldsymbol{W}_j^\ell \phi(\mathcal{N}^{\ell-1}(\mathbf{x})) + \boldsymbol{b}_j^\ell, \qquad 1 \le j \le N_\ell$$

that we have

$$E\left[\mathcal{N}_j^\ell(\mathbf{x})^2 | \mathcal{N}^{\ell-1}(\mathbf{x})\right] = \sigma_\ell^2 \sum_{t \neq k_j^\ell} \phi(\mathcal{N}_t^{\ell-1}(\mathbf{x}))^2 + \mu'_{\ell,2} \phi(\mathcal{N}_{k_j^\ell}^{\ell-1}(\mathbf{x}))^2.$$

Since k_j^ℓ is randomly uniformly selected, by taking the expectation with respect to k_j^ℓ, we have

$$\mathbb{E}\left[\mathcal{N}_j^\ell(\mathbf{x})^2 | \mathcal{N}^{\ell-1}(\mathbf{x})\right] = \frac{\sigma_\ell^2 N_\ell + \mu'_{\ell,2}}{N_\ell + 1} \left(1 + \|\phi(\mathcal{N}^{\ell-1}(\mathbf{x}))\|^2\right).$$

Since the above is independent of j, and $q^\ell(\mathbf{x}) = \|\mathcal{N}^\ell(\mathbf{x})\|^2/N_\ell$,

$$\mathbb{E}\left[q^\ell(\mathbf{x}) | \mathcal{N}^{\ell-1}(\mathbf{x})\right] = \frac{\sigma_\ell^2 N_\ell + \mu'_{\ell,2}}{N_\ell + 1} \left(1 + \|\phi(\mathcal{N}^{\ell-1}(\mathbf{x}))\|^2\right). \tag{A.8}$$

At a fixed ℓ and for each $t = 1, \cdots, N_\ell$, let $\mathbf{v}^{\ell,t} = [\mathbf{W}_t^\ell, \mathbf{b}_t^\ell] \in \mathbb{R}^{N_{\ell-1}+1}$ be a random vector such that $\mathbf{v}_{-k_t}^{\ell,t} \sim N(0, \sigma_\ell^2 \boldsymbol{I}_{N_{\ell-1}})$ and $\mathbf{v}_{k_t}^{\ell,t} \sim \mathrm{P}_\ell$. Then $\mathcal{N}_t^\ell(\mathbf{x}) = \langle \mathbf{v}^{\ell,t}, \mathbf{n}^\ell(\mathbf{x}) \rangle$. Given $\mathcal{N}^{\ell-1}(\mathbf{x})$, assuming $(\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x})) > 0$, one can view $\mathcal{N}_t^\ell(\mathbf{x})$ as

$$\mathcal{N}_t^\ell(\mathbf{x}) = \mathbf{n}_{k_t}^{\ell-1}(\mathbf{x}) \left(\sigma_{\ell,k_t} Z + X_\ell\right), \qquad Z \sim N(0,1) \text{ and } X_\ell \sim \mathrm{P}_\ell.$$

Thus $\phi(\mathcal{N}_t^\ell(\mathbf{x}))^2 = \left(\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x})\right)^2 \phi\left(\sigma_{\ell,k_t} Z + X_\ell\right)^2$ and we obtain

$$
\begin{aligned}
&\frac{1}{(\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x}))^2} E\left[\phi(\mathcal{N}_t^\ell(\mathbf{x}))^2 | \mathcal{N}^{\ell-1}(\mathbf{x})\right] = E\left[\phi\left(\sigma_{\ell,k_t} Z + X_\ell\right)^2 | \mathcal{N}^{\ell-1}(\mathbf{x})\right] \\
&= \int_{-\infty}^{M} \int_{y}^{M} (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y) dy \\
&= \int_{-\infty}^{0} \int_{0}^{M} (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y) dy + \int_{0}^{M} \int_{y}^{M} (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y) dy \\
&= \int_{-\infty}^{0} \int_{0}^{M} \left(x^2 + y^2 - 2xy\right) dF_P(x) f_Y^{\ell,k_t}(y) dy + \int_{0}^{M} \int_{y}^{M} (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y) dy \\
&= \int_{-\infty}^{0} \left(\mu'_{\ell,2} + y^2 - 2\mu'_{\ell,1} y\right) f_Y^{\ell,k_j}(y) dy + \int_{0}^{M} \int_{y}^{M} (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y) dy \\
&= \frac{1}{2}\left(\mu'_{\ell,2} + \tilde{\sigma}_{\ell,k_t}^2\right) + \mu'_{\ell,1}\sqrt{\frac{2}{\pi}}\tilde{\sigma}_{\ell,k_t} + \int_{0}^{M} \int_{y}^{M} (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y) dy.
\end{aligned}
$$

By multiplying $(\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x}))^2$ in the both sides, we have

$$
\begin{aligned}
E[\phi(\mathcal{N}_t^\ell(\mathbf{x}))^2 | \mathcal{N}^{\ell-1}(\mathbf{x})] = \frac{1}{2}\Big((\mu'_{\ell,2} - \sigma_\ell^2)(\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x}))^2 + (\|\phi(\mathcal{N}^{\ell-1}(\mathbf{x}))\|^2 + 1)\sigma_\ell^2\Big) \\
+ (\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x}))^2 \int_{0}^{M} \int_{y}^{M} (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y) dy \\
+ \mu'_{\ell,1}\sigma_\ell \sqrt{\frac{2}{\pi}} \mathbf{n}_{k_t}^{\ell-1}(\mathbf{x}) \|\mathbf{n}_{-k_t}^{\ell-1}(\mathbf{x})\|.
\end{aligned}
$$

Since k_t is randomly selected, by taking expectation w.r.t. k_j, we have

$$
\begin{aligned}
\mathbb{E}[\phi(\mathcal{N}_t^\ell(\mathbf{x}))^2 | \mathcal{N}^{\ell-1}(\mathbf{x})] = \frac{(1 + \|\phi(\mathcal{N}^{\ell-1}(\mathbf{x}))\|^2)}{2}\left(\frac{\mu'_{\ell,2} - \sigma_\ell^2}{N_{\ell-1}+1} + \sigma_\ell^2\right) \\
+ \sum_{k_t=1}^{N_{\ell-1}+1} \frac{(\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x}))^2}{N_{\ell-1}+1} \int_{0}^{M} \int_{y}^{M} (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y) dy \\
+ \mu'_{\ell,1}\sqrt{\frac{2}{\pi}}\sigma_\ell \sum_{k_t=1}^{N_{\ell-1}+1} \frac{\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x}) \|\mathbf{n}_{-k_t}^{\ell-1}(\mathbf{x})\|}{N_{\ell-1}+1}.
\end{aligned}
\tag{A.9}
$$

By the Cauchy-Schwarz inequality, the third term in the right hand side of Equation A.9 can be bounded by

$$\mu'_{\ell,1}\sqrt{\frac{2}{\pi}}\sigma_\ell \sum_{k_t=1}^{N_{\ell-1}+1} \frac{\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x})\|\mathbf{n}_{-k_t}^{\ell-1}(\mathbf{x})\|}{N_{\ell-1}+1} \leq \mu'_{\ell,1}\sqrt{\frac{2}{\pi}}\frac{\sigma_w(1+\|\phi(\mathcal{N}^{\ell-1}(\mathbf{x}))\|^2)}{N_{\ell-1}+1}.$$

Let $I_{\ell,k_t} = \int_0^M \int_y^M (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y)dy$. It then follows from Lemma A.6.1 that $I_{\ell,k} \leq \mu'_{\ell,2}/2$ for any k. Thus the second term in the right hand side of Equation A.9 can be bounded by

$$\sum_{k_t=1}^{N_{\ell-1}+1} \frac{(\mathbf{n}_{k_t}^{\ell-1}(\mathbf{x}))^2}{N_{\ell-1}+1} \int_0^M \int_y^M (x-y)^2 dF_P(x) f_Y^{\ell,k_t}(y)dy \leq \frac{\mu'_{\ell,2}}{2}\frac{\left(1+\|\phi(\mathcal{N}^{\ell-1}(\mathbf{x}))\|^2\right)}{N_{\ell-1}+1}$$

Therefore, we obtain

$$E[\phi(\mathcal{N}_t^\ell(\mathbf{x}))^2|\mathcal{N}^{\ell-1}(\mathbf{x})] \leq C_\ell \frac{(1+\|\phi(\mathcal{N}^{\ell-1}(\mathbf{x}))\|^2)}{2}$$

where

$$C_\ell = \frac{\mu'_{\ell,2}-\sigma_\ell^2}{N_{\ell-1}+1} + \sigma_\ell^2 + \frac{2\sqrt{2}\mu'_{\ell,1}\sigma_w}{(N_{\ell-1}+1)\sqrt{\pi}} + \frac{\mu'_{\ell,2}}{N_{\ell-1}+1}.$$

Since C_ℓ is independent of t,

$$E[\|\phi(\mathcal{N}^\ell(\mathbf{x}))\|^2|\mathcal{N}^{\ell-1}(\mathbf{x})] \leq N_\ell C_\ell \frac{(1+\|\phi(\mathcal{N}^{\ell-1}(\mathbf{x}))\|^2)}{2}.$$

It then follows from Equation A.8,

$$1+\|\phi(\mathcal{N}^{\ell-1}(\mathbf{x}))\|^2 = \frac{N_\ell+1}{\sigma_\ell^2 N_\ell + \mu'_{\ell,2}} E\left[q^\ell(\mathbf{x})|\mathcal{N}^{\ell-1}(\mathbf{x})\right]$$

that

$$E[\|\phi(\mathcal{N}^{\ell}(\mathbf{x}))\|^2|\mathcal{N}^{\ell-1}(\mathbf{x})] \leq \frac{CN_{\ell}(N_{\ell}+1)}{2(\sigma_{\ell}^2 N_{\ell}+\mu'_{\ell,2})} E\left[q^{\ell}(\mathbf{x})|\mathcal{N}^{\ell-1}(\mathbf{x})\right].$$

Thus we have

$$\begin{aligned} E\left[q^{\ell+1}(\mathbf{x})|\mathcal{N}^{\ell-1}(\mathbf{x})\right] &= \frac{\sigma_{\ell+1}^2 N_{\ell+1}+\mu'_{\ell+1,2}}{N_{\ell+1}+1}\left(1+E[\|\phi(\mathcal{N}^{\ell}(\mathbf{x}))\|^2|\mathcal{N}^{\ell-1}(\mathbf{x})]\right) \\ &\leq \sigma_{b,\ell}^2 + \frac{\mathcal{A}_{\ell,upp}}{2} E\left[q^{\ell}(\mathbf{x})|\mathcal{N}^{\ell-1}(\mathbf{x})\right] \end{aligned}$$

where

$$\sigma_{b,\ell}^2 = \frac{\sigma_{\ell+1}^2 N_{\ell+1}+\mu'_{\ell+1,2}}{N_{\ell+1}+1}, \qquad \mathcal{A}_{\ell,upp} = \frac{\sigma_{b,\ell}^2 N_{\ell}(N_{\ell}+1)}{\sigma_{\ell}^2 N_{\ell}+\mu'_{\ell,2}} C_{\ell}.$$

By taking expectation with respect to $\mathcal{N}^{\ell-1}(\mathbf{x})$, we obtain

$$E[q^{\ell+1}(\mathbf{x})] \leq \frac{\mathcal{A}_{\ell,upp}}{2} E[q^{\ell}(\mathbf{x})] + \sigma_{b,\ell}^2$$

The lower bound can be obtained by dropping the second and the third terms in Equation A.9. Thus

$$\frac{\mathcal{A}_{\ell,low}}{2} E[q^{\ell}(\mathbf{x})] + \sigma_{b,\ell}^2 \leq E[q^{\ell+1}(\mathbf{x})] \leq \frac{\mathcal{A}_{\ell,upp}}{2} E[q^{\ell}(\mathbf{x})] + \sigma_{b,\ell}^2$$

where

$$\mathcal{A}_{\ell,low} = \frac{\sigma_{\ell+1}^2 N_{\ell+1}+\mu'_{\ell+1,2}}{N_{\ell+1}+1} \frac{N_{\ell}+1}{\sigma_{\ell}^2 N_{\ell}+\mu'_{\ell,2}} \left(\frac{\mu'_{\ell,2} N_{\ell}-\sigma_w^2}{N_{\ell-1}+1}+\sigma_w^2\right).$$

□

A.7 Randomized asymmetric initialization

```python
import numpy as np


def RAI(fan_in, fan_out):
    """Randomized asymmetric initializer.
    It draws samples using RAI where fan_in is the number of input units in the weight tensor and fan_out is the number of output units in the weight tensor.
    """
    V = np.random.randn(fan_out, fan_in + 1) * 0.6007 / fan_in ** 0.5
    for j in range(fan_out):
        k = np.random.randint(0, high=fan_in + 1)
        V[j, k] = np.random.beta(2, 1)
    W = V[:, :-1].T
    b = V[:, -1]
    return W.astype(np.float32), b.astype(np.float32)
```

Appendix

B

Quantifying the Generalization Error in Deep Learning in terms of Data Distribution and Neural Network Smoothness

B.1 Example of topology

Example B.1.1. *Given*

$$T = 2^{\{1,2,3\}} \backslash \{\varnothing\} = \{\ \{1\},\ \{2\},\ \{3\},\ \{1,2\},\ \{1,3\},\ \{2,3\},\ \{1,2,3\}\ \},$$

$$\begin{aligned} U = \{\ &\{\{1\},\{1,2\},\{1,3\},\{1,2,3\}\},\ \{\{2\},\{2,1\},\{2,3\},\{1,2,3\}\},\\ &\{\{3\},\{3,1\},\{3,2\},\{1,2,3\}\},\ \{\{1,2\},\{1,2,3\}\},\\ &\{\{1,3\},\{1,2,3\}\},\ \{\{2,3\},\{1,2,3\}\},\ \{\{1,2,3\}\}\ \}, \end{aligned}$$

then τ_T is the topology generated by U.

In this example, $\{\{1\},\{1,2\},\{1,3\},\{1,2,3\}\}$ is an open set, since it consists of all elements containing label $\{1\}$, and $\{\{2,3\},\{1,2,3\}\}$ is also an open set with common part $\{2,3\}$. Besides open sets from base U, $\{\{1\},\{1,2\},\{1,3\},\{2,3\},\{1,2,3\}\}$ is still an open set as the union of the two shown above.

B.2 Proof of Proposition 3.2.3

Proof. We use the proof by contradiction. Assume that the result does not hold, then

$$\exists \{x_n\}, \{y_n\} \subseteq D,\ s.t.\ \|x_n - y_n\| \to 0 (n \to \infty), \text{ and } tag(x_n) \cap tag(y_n) = \varnothing.$$

Because D is compact, there exist $x^* \in D$ and a subsequence $\{x_{k_n}\}$ of $\{x_n\}$ such that $x_{k_n} \to x^*$. As $\|x_n - y_n\| \to 0$, also $y_{k_n} \to x^*$. Choose any $i_0 \in tag(x^*)$, then

there exists a sufficient large k_{n_0} such that $i_0 \in tag(x_{k_{n_0}})$, $i_0 \in tag(y_{k_{n_0}})$. Therefore $tag(x_{k_{n_0}}) \cap tag(y_{k_{n_0}}) \neq \varnothing$, which contradicts the assumption. □

B.3 Proof of Proposition 3.2.4

Proof. Let δ_0 as defined in this proposition. For any two different points x, y with distance less than δ_0, either $tag(x) = tag(y)$, or at least one of the two is a full label point, in both cases $tag(x) \cap tag(y) \neq \varnothing$. For any $\delta > \delta_0$, according to the definition of δ_0, there exist two points x_0, y_0 satisfying

$$\|x_0 - y_0\| < \delta, \quad tag(x_0) \cap tag(y_0) = \varnothing.$$

The two facts imply that δ_0 is the supremum of δ satisfying Proposition 3.2.3. □

B.4 Proof of Proposition 3.2.7

Proof. According to the definition,

$$\begin{aligned}
\sqrt{d}\rho_{\mathcal{T}} &= \int_0^{\sqrt{d}} h_{\mathcal{T}}^{\mu}(t)dt \\
&= \int_0^{r} h_{\mathcal{T}}^{\mu}(t)dt + \int_r^{\sqrt{d}} h_{\mathcal{T}}^{\mu}(t)dt \\
&\leq r \cdot h_{\mathcal{T}}^{\mu}(r) + (\sqrt{d} - r) \cdot 1 \\
&= \sqrt{d} - r(1 - h_{\mathcal{T}}^{\mu}(r)),
\end{aligned}$$

thus

$$h_{\mathcal{T}}^{\mu}(r) \geq 1 - \frac{\sqrt{d}}{r}(1 - \rho_{\mathcal{T}}).$$

□

B.5 Estimate of total cover

In this section, we estimate the TC by the number of samples in the training set. The notations, such as D, d, P, μ, $\rho_{\mathcal{T}}$, as well as training set

$$\mathcal{T} = \{x_1, x_2, \ldots, x_n\} \subseteq D$$

are the same as before. Note that samples in $\mathcal{T}$ are drawn according to μ. Before presenting the analysis, we first collect the following auxiliary notions and results (**Definitions B.5.1-B.5.4, Theorem B.5.5**) which are easily found in [150] (Definitions 14.1-14.3, Definition 14.5, and Theorem 14.8):

Definition B.5.1. *A range space is a pair $(X, \mathcal{R})$ where:*

1. *X is a (finite or infinite) set of points;*
2. *$\mathcal{R}$ is a family of subsets of X, called ranges.*

Definition B.5.2. *Let $(X, \mathcal{R})$ be a range space and let $Y \subseteq X$. The projection of $\mathcal{R}$ on Y is*

$$\mathcal{R}_Y = \{R \cap Y | R \in \mathcal{R}\}.$$

Definition B.5.3. *Let $(X, \mathcal{R})$ be a range space. A set $Y \subseteq X$ is shattered by $\mathcal{R}$ if $|\mathcal{R}_Y| = 2^{|Y|}$. The Vapnik-Chervonenkis (VC) dimension of a range space $(X, \mathcal{R})$ is the*

maximum cardinality of a set $Y \subseteq X$ that is shattered by $\mathcal{R}$. If there are arbitrarily large finite sets that are shattered by $\mathcal{R}$, then the VC dimension is infinite.

Definition B.5.4. *Let $(X, \mathcal{R})$ be a range space, and let $\mathcal{F}$ be a probability distribution on X. A set $N \subseteq X$ is an $\epsilon - net$ for X with respect to $\mathcal{F}$ if for any set $R \in \mathcal{R}$ such that $Pr_{\mathcal{F}}(R) \geq \epsilon$, the set R contains at least one point from N, i.e.,*

$$\forall R \in \mathcal{R}, Pr_{\mathcal{F}}(R) \geq \epsilon \Rightarrow R \cap N \neq \emptyset.$$

Theorem B.5.5. *Let $(X, \mathcal{R})$ be a range space with VC dimension d_{vc} and let $\mathcal{F}$ be a probability distribution on X. For any $0 < \eta, \epsilon \leq 1/2$, there is an*

$$m = O\left(\frac{d_{vc}}{\epsilon} \ln \frac{d_{vc}}{\epsilon} + \frac{1}{\epsilon} \ln \frac{1}{\eta}\right)$$

such that a random sample from $\mathcal{F}$ of size greater than or equal to m is an $\epsilon - net$ for X with probability at least $1 - \eta$.

Now let

$$B_D = \{B(x, r) \cap D | x \in \mathbb{R}^d, r > 0\},$$

we first show (D, B_D) is a range space with VC dimension $d + 1$.

Lemma B.5.6. *The VC dimension of range space (D, B_D) is $d + 1$.*

Proof. The proof can be found in [58]. □

Set

$$B_D^{\epsilon}(r) = \{A \in B_D | A = B(x, s) \cap D, 0 < s \leq r, \mu(A) \geq \epsilon\},$$

and

$$\varrho(\epsilon) = \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} \mu\left(\bigcup_{A \in B_D^{\epsilon}(\frac{1}{2}r)} A \right) dr,$$

we have the following lemmas.

Lemma B.5.7. $\bigcup_{A \in B_D^{\epsilon}(\frac{1}{2}r)} A \subseteq D \cap \bigcup_{x_i \in \mathcal{T}} B(x_i, r)$ *when* $\mathcal{T}$ *is an* $\epsilon - net$ *for* (D, B_D).

Proof. For any $x \in \bigcup_{A \in B_D^{\epsilon}(\frac{1}{2}r)} A$, assume that $x \in A^* = B(x^*, s^*) \cap D, s^* \leq \frac{1}{2}r, \mu(A^*) \geq \epsilon$. Since $\mathcal{T}$ is an $\epsilon - net$ and $\mu(A^*) \geq \epsilon$, we know $\mathcal{T} \cap A^* \neq \emptyset$. Thus there exists $x_t \in \mathcal{T}$ such that $\|x_t - x^*\| < s^* \leq \frac{1}{2}r$. Therefore

$$\|x - x_t\| \leq \|x - x^*\| + \|x^* - x_t\| < \frac{1}{2}r + \frac{1}{2}r = r.$$

The above inequality shows that

$$x \in D \cap B(x_t, r) \subseteq D \cap \bigcup_{x_i \in \mathcal{T}} B(x_i, r).$$

□

Lemma B.5.8. $\lim_{\epsilon \to 0} \varrho(\epsilon) = 1$.

Proof. For any positive decreasing sequence $\{\epsilon_i\}$ which satisfies $\lim_{i \to \infty} \epsilon_i = 0$, it leads to an incremental chain

$$\left(\bigcup_{A \in B_D^{\epsilon_i}(\frac{1}{2}r)} A \right) \subseteq \left(\bigcup_{A \in B_D^{\epsilon_{i+1}}(\frac{1}{2}r)} A \right), \quad \lim_{i \to \infty} \left(\bigcup_{A \in B_D^{\epsilon_i}(\frac{1}{2}r)} A \right) = \widetilde{D} \subseteq D,$$

where $\widetilde{D} = \bigcup_{A \in B_D^{+}(\frac{1}{2}r)} A$ for $B_D^{+}(\frac{1}{2}r) = \{A \in B_D | A = B(x, s) \cap D, 0 < s \leq \frac{1}{2}r, \mu(A) > 0\}$. Let us consider a series of open balls $\{B_i\}$ of radius at most $\frac{1}{2}r$ that cover D,

and we divide them into two parts $\{B_i\} = \{B_i^+\} \cup \{B_i^0\}$ such that $\mu(B_i^+ \cap D) > 0$ and $\mu(B_i^0 \cap D) = 0$. Then $\mu(\widetilde{D}) \geq \mu(D \cap \bigcup_i B_i^+) \geq \mu(D) - \mu(D \cap \bigcup_i B_i^0) = 1 - 0 = 1$, and thus $\mu(\widetilde{D}) = 1$. Therefore, we have $\lim_{\epsilon \to 0} \mu(\bigcup_{A \in B_D^\epsilon(\frac{1}{2}r)} A) = \mu(\widetilde{D}) = 1$.

Since

$$\lim_{\epsilon \to 0} \mu \left(\bigcup_{A \in B_D^\epsilon(\frac{1}{2}r)} A \right) = 1 \; and \; \mu \left(\bigcup_{A \in B_D^\epsilon(\frac{1}{2}r)} A \right) \leq 1,$$

by dominated convergence theorem, we have

$$\begin{aligned}
\lim_{\epsilon \to 0} \varrho(\epsilon) &= \lim_{\epsilon \to 0} \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} \mu \left(\bigcup_{A \in B_D^\epsilon(\frac{1}{2}r)} A \right) dr \\
&= \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} \lim_{\epsilon \to 0} \mu \left(\bigcup_{A \in B_D^\epsilon(\frac{1}{2}r)} A \right) dr \\
&= \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} 1 dr \\
&= 1.
\end{aligned}$$

□

According to the aforementioned lemmas, we deduce the following theorem.

Theorem B.5.9. *Let $\mathcal{T} = \{x_1, x_2, \ldots, x_n\}$ be the training set drawn according to μ, then for any $0 < \eta, \epsilon \leq \frac{1}{2}$, there exists an*

$$m = O\left(\frac{d+1}{\epsilon} \ln \frac{d+1}{\epsilon} + \frac{1}{\epsilon} \ln \frac{1}{\eta} \right)$$

such that

$$\rho_{\mathcal{T}} \geq \varrho(\epsilon)$$

holds with probability at least $1 - \eta$ when $n \geq m$. Note that $\varrho(\epsilon) \to 1$ when $\epsilon \to 0$.

Proof. Theorem B.5.5 shows that $\mathcal{T}$ is an $\epsilon - net$ for range space (D, B_D) with probability at least $1 - \eta$ when $n \geq m$. By lemma B.5.7, we have

$$\begin{aligned}
\rho_{\mathcal{T}} &= \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} \mu \left(D \cap \bigcup_{x_i \in \mathcal{T}} B(x_i, r) \right) dr \\
&\geq \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} \mu \left(\bigcup_{A \in B_D^{\epsilon}(\frac{1}{2}r)} A \right) dr \\
&= \varrho(\epsilon).
\end{aligned}$$

□

From this theorem, we know that a large number of samples lead to a sufficiently large $\rho_{\mathcal{T}}$ with a high probability.

In the previous sections, there is an assumption that our obtained training set is of single label, so we will naturally consider this special case in the sequel.

Denote

$$D_{sin} = \{x \in D |\ tag(x) \in \{\{1\}, \{2\}, \cdots, \{K\}\}\ \},$$

$$B_{D_{sin}} = \{B(x, r) \cap D_{sin} | x \in \mathbb{R}^d, r > 0\},$$

$$\mu_{sin}(A) = \frac{\mu(A \cap D_{sin})}{\mu(D_{sin})},$$

and $(D_{sin}, B_{D_{sin}})$ is a range space with VC dimension at most $d + 1$. Let

$$\mathcal{T} = \{x_1, \cdots, x_n\} \subseteq D_{sin},$$

that is, the samples in $\mathcal{T}$ are drawn according to μ_{sin}. As before, denote

$$B^{\epsilon}_{D_{sin}}(r) = \{A \in B_D | A = B(x,s) \cap D, 0 < s \leq r, \mu_{sin}(A) \geq \epsilon\},$$

$$\varrho_{sin}(\epsilon) = \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} \mu \left(\bigcup_{A \in B^{\epsilon}_{D_{sin}}(\frac{1}{2}r)} A \right) dr.$$

We have the following lemmas.

Lemma B.5.10. $\bigcup_{A \in B^{\epsilon}_{D_{sin}}(\frac{1}{2}r)} A \subseteq D \cap \bigcup_{x_i \in \mathcal{T}} B(x_i, r)$ *when* $\mathcal{T}$ *is an* ϵ*-net for* $(D_{sin}, B_{D_{sin}})$.

Lemma B.5.11. $\lim_{\epsilon \to 0} \varrho_{sin}(\epsilon) = c_{sin}$, *here*

$$c_{sin} = \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} \mu \left(\bigcup_{A \in C_{D_{sin}}(\frac{1}{2}r)} A \right) dr,$$

$$C_{D_{sin}}(r) = \{A \in B_D | A = B(x,s) \cap D, 0 < s \leq r, \mu_{sin}(A) > 0\}.$$

From these two lemmas we deduce the following theorem.

Theorem B.5.12. *Let* $\mathcal{T} = \{x_1, x_2, \ldots, x_n\}$ *be the training set drawn according to* μ_{sin}, *then for any* $0 < \eta, \epsilon \leq \frac{1}{2}$, *there exists an*

$$m = O\left(\frac{d+1}{\epsilon} \ln \frac{d+1}{\epsilon} + \frac{1}{\epsilon} \ln \frac{1}{\eta} \right)$$

such that

$$\rho_{\mathcal{T}} \geq \varrho_{sin}(\epsilon)$$

holds with probability at least $1 - \eta$ *when* $n \geq m$. *Note that* $\varrho_{sin}(\epsilon) \to c_{sin}$ *when* $\epsilon \to 0$.

The proofs for Lemmas B.5.10-B.5.11 and Theorem B.5.12 are very similar to

those for Lemmas B.5.7-B.5.8 and Theorem B.5.9, respectively. We omit them here. It is noteworthy that c_{sin} is intuitively very close to 1, even equal to 1. At worst, c_{sin} is at least greater than $\mu(D_{sin})$ which may be quite large in practice, and the proof is similar to what we show in B.5.8.

B.6 Proof of Proposition 3.2.9

Proof. Let $\mathcal{T}$ be the training set and λ be a positive constant greater than 1, $\widetilde{\mathcal{T}} = \mathcal{T}/\lambda$, $\widetilde{\mu}(A) = \mu(\lambda A)$, then

$$\begin{aligned}
1 - \rho_{\widetilde{\mathcal{T}}} &= \frac{1}{\sqrt{d}} \int_0^{\sqrt{d}} (1 - h^{\widetilde{\mu}}_{\widetilde{\mathcal{T}}}(r)) dr \\
&= \frac{1}{\sqrt{d}} \int_0^{\infty} (1 - h^{\widetilde{\mu}}_{\widetilde{\mathcal{T}}}(r)) dr \\
&= \frac{1}{\sqrt{d}} \int_0^{\infty} (1 - h^{\mu}_{\mathcal{T}}(\lambda r)) dr \\
&= \frac{1}{\lambda\sqrt{d}} \int_0^{\infty} (1 - h^{\mu}_{\mathcal{T}}(r)) dr \\
&= (1 - \rho_{\mathcal{T}})/\lambda.
\end{aligned}$$

For the same reason,

$$\begin{aligned}
CD(\widetilde{\mathcal{T}}) &= \frac{1}{K(K-1)} \sum_{i \neq j} (1 - \rho(\widetilde{\mathcal{T}}_i, \widetilde{\mu}_j)) - \frac{1}{K} \sum_i (1 - \rho(\widetilde{\mathcal{T}}_i, \widetilde{\mu}_i)) \\
&= \frac{1}{\lambda K(K-1)} \sum_{i \neq j} (1 - \rho(\mathcal{T}_i, \mu_j)) - \frac{1}{\lambda K} \sum_i (1 - \rho(\mathcal{T}_i, \mu_i)) \\
&= CD(\mathcal{T})/\lambda,
\end{aligned}$$

therefore

$$CC(\widetilde{\mathcal{T}}) = CC(\mathcal{T}).$$

□

B.7 Proof of Proposition 3.3.1

Proof. Denote by $\widetilde{\mathcal{T}} = \{x_i \in \mathcal{T} \mid f \text{ is } c_1\text{−accurate at } x_i\}$, $\delta = \min(\delta_0, \delta_f(c_1 - c_2))$. For any $x_i \in \widetilde{\mathcal{T}}$, choose $k_i \in \{1, \cdots, K\}$ such that $tag(x_i) = \{k_i\}$. From the definition of $\widetilde{\mathcal{T}}$ we know that $f_{k_i}(x_i) > c_1$.

For any $x \in B(x_i, \delta) \cap D$, from Proposition 3.2.3 we know that $tag(x) \cap tag(x_i) \neq \varnothing$, and hence $k_i \in tag(x)$. On the other hand, $\|f(x) - f(x_i)\|_\infty < c_1 - c_2$, so we have $|f_{k_i}(x) - f_{k_i}(x_i)| < c_1 - c_2$, therefore

$$f_{k_i}(x) > f_{k_i}(x_i) - (c_1 - c_2) > c_1 - (c_1 - c_2) = c_2,$$

which means that f is c_2−accurate at x, that is to say

$$H^f_{c_2} \supseteq D \cap \bigcup_{x_i \in \widetilde{\mathcal{T}}} B(x_i, \delta).$$

Then

$$\begin{aligned} p_{c_2}(f) &= \mu(H^f_{c_2}) \\ &\geq \mu\left(D \cap \bigcup_{x_i \in \widetilde{\mathcal{T}}} B(x_i, \delta)\right) \\ &= h^\mu_{\widetilde{\mathcal{T}}}(\delta) \\ &\geq 1 - \frac{\sqrt{d}}{\delta}(1 - \rho_{\widetilde{\mathcal{T}}}) \\ &= 1 - \frac{\sqrt{d}}{\delta}(1 - p^{\mathcal{T}}_{c_1}\rho_{\mathcal{T}}). \end{aligned}$$

The second inequality can be derived from Proposition 3.2.7. □

B.8 Proof of Theorem 3.3.2

Proof. Define $\delta := \min(\delta_0, \delta_f(e^{-L_f^{max}} - c))$. Note that if $x_i \in \mathcal{T}$, and $x \in B(x_i, \delta) \cap D$, then $\|x - x_i\| < \delta_0$, and hence $tag(x) \cap tag(x_i) \neq \emptyset$, so that $k_i \in tag(x)$.

Because of

$$\|x - x_i\| < \delta_f(e^{-L_f^{max}} - c),$$

we have

$$\|f(x) - f(x_i)\|_\infty < e^{-L_f^{max}} - c,$$

so that $|f_{k_i}(x) - f_{k_i}(x_i)| < e^{-L_f^{max}} - c$. Therefore

$$f_{k_i}(x) > f_{k_i}(x_i) - e^{-L_f^{max}} + c \geq e^{-L_f^{max}} - e^{-L_f^{max}} + c = c,$$

so that f is c-accurate at x. Overall we obtain

$$\begin{aligned} p_c(f) &= \mu(H_c^f) \\ &\geq \mu\left(D \cap \bigcup_{x_i \in \mathcal{T}} B(x_i, \delta)\right) \\ &= h_{\mathcal{T}}^{\mu}(\delta) \\ &\geq 1 - \frac{\sqrt{d}}{\delta}(1 - \rho_{\mathcal{T}}). \end{aligned}$$

The second inequality can be derived from Proposition 3.2.7. □

B.9 Proof of Theorem 3.3.4

Proof. We will prove it by contradiction. Consider two points x_1 and x_2 with different labels, and $\|x_1 - x_2\| = \delta_{\mathcal{T}}$.

If $\delta_f(e^{-L_f^{max}} - c) > \delta_{\mathcal{T}}/2$, then by the definition of $\delta_f(e^{-L_f^{max}} - c)$, $\|f(\frac{x_1+x_2}{2}) - f(x_1)\|_\infty < e^{-L_f^{max}} - c$, since $\|\frac{x_1+x_2}{2} - x_1\| = \|\frac{x_2-x_1}{2}\| = \delta_{\mathcal{T}}/2$. Similarly, $\|f(\frac{x_1+x_2}{2}) - f(x_2)\|_\infty < e^{-L_f^{max}} - c$. Then we have

$$\|f(x_1) - f(x_2)\|_\infty < 2(e^{-L_f^{max}} - c) \leq 2e^{-L_f^{max}} - 1.$$

On the other hand, by the definition of L_f^{max}, $-\ln f_{k_1}(x_1) \leq L_f^{max}$ and $-\ln f_{k_2}(x_2) \leq L_f^{max}$, so $f_{k_1}(x_1) \geq e^{-L_f^{max}}$, $f_{k_2}(x_2) \geq e^{-L_f^{max}}$ and $f_{k_1}(x_2) \leq 1 - f_{k_2}(x_2) \leq 1 - e^{-L_f^{max}}$, therefore

$$\begin{aligned}
\|f(x_1) - f(x_2)\|_\infty &\geq f_{k_1}(x_1) - f_{k_1}(x_2) \\
&\geq e^{-L_f^{max}} - (1 - e^{-L_f^{max}}) \\
&= 2e^{-L_f^{max}} - 1.
\end{aligned}$$

□

B.10 Proof of Theorem 3.3.5

Proof. From Theorem 3.3.2 and assumption we know that

$$\begin{aligned} p(f) \geq p_{0.5}(f) &> 1 - \frac{\sqrt{d}}{\min(\delta_0, \delta_f(e^{-L_f^{max}} - 0.5))}(1 - \rho_{\mathcal{T}}) \\ &\geq 1 - \frac{\sqrt{d}}{\min(\delta_0, \kappa\delta_{\mathcal{T}})}(1 - \rho_{\mathcal{T}}) \\ &\geq 1 - \frac{\sqrt{d}}{\kappa\delta_0}(1 - \rho_{\mathcal{T}}), \end{aligned}$$

which implies $\lim_{\rho_{\mathcal{T}} \to 1} p(f) = 1$. Note that κ is less than 0.5 and δ_0 is only determined by the classification problem itself. The above inequality is easy to convert into form (ii). □

B.11 Detailed information of data and parameters for training

First, we list the information concerning the data selection.

1. MNIST: Last 55000 samples of the training set for training and all the 10000 samples of the test set for testing.
2. CIFAR-10: First 49000 samples of the training set for training and all the 10000 samples of the test set for testing.
3. CIFAR-100: First 49000 samples of the training set for training and all the 10000 samples of the test set for testing.

4. COIL-20: 1200 samples whose end numbers of the figure names are not multiples of 6 for training and 240 samples whose end numbers are multiples of 6 for testing.

5. COIL-100: 6000 samples whose end numbers of the figure names are not multiples of 30 for training and 1200 samples whose end numbers are multiples of 30 for testing.

6. SVHN: First 50000 samples of the training set for training and first 10000 samples of the test set for testing.

Parameters for networks are listed in Table B.1.

Table B.1: The results in this table were obtained as follows: We used the ReLU as the activation function and Adam as the optimizer. Furthermore, we chose several different network structures and learning rates. We then trained the networks for 10000 iterations with a batch size of 300, and selected the best test error as the final result for each training procedure. Each of the columns *i*-*xii* stands for a specific choice of network architecture and learning rate, as described in Table B.2

Data Set	Version	*i*	*ii*	*iii*	*iv*	*v*	*vi*	*vii*	*viii*	*ix*	*x*	*xi*	*xii*	Best Error
MNIST	Original	.02	.02	.05	.02	.02	.04	.02	.02	.04	.01	.02	.03	.01
CIFAR-10	Original	.47	.46	.52	.48	.46	.51	.47	.45	.50	.47	.45	.49	.45
CIFAR-10	Grey	.55	.55	.63	.55	.54	.62	.54	.53	.61	.54	.53	.59	.53
CIFAR-10	Conv	.18	.18	.19	.19	.18	.19	.18	.18	.18	.18	.18	.18	.18
SVHN	Original	.80	.59	.49	.80	.73	.60	.80	.69	.51	.80	.72	.64	.49
SVHN	Grey	.80	.64	.56	.80	.76	.66	.80	.64	.58	.80	.75	.66	.56
SVHN	Conv	.27	.23	.23	.31	.24	.23	.31	.24	.23	.69	.25	.24	.23
CIFAR-100	Original(coarse)	.64	.64	.69	.64	.63	.68	.63	.62	.67	.63	.62	.66	.62
CIFAR-100	Grey(coarse)	.74	.73	.79	.74	.73	.78	.74	.72	.78	.74	.72	.77	.72
CIFAR-100	Conv(coarse)	.40	.41	.45	.41	.41	.44	.40	.41	.44	.41	.40	.43	.40
COIL-20	Original	.08	.05	.05	.07	.06	.05	.08	.05	.05	.07	.05	.03	.03
CIFAR-100	Original(fine)	.75	.75	.82	.75	.74	.81	.74	.73	.80	.75	.73	.79	.73
CIFAR-100	Grey(fine)	.83	.83	.90	.83	.83	.90	.82	.81	.88	.83	.81	.87	.81
CIFAR-100	Conv(fine)	.53	.54	.64	.53	.54	.63	.52	.52	.61	.53	.52	.59	.52
COIL-100	Original	.02	.01	.01	.03	.02	.01	.03	.02	.01	.02	.02	.01	.01

Table B.2: Detailed setup for each case.

Structure \ Learning Rate	10^{-3}	10^{-4}	10^{-5}
[Input 256 256 Output]	i	ii	iii
[Input 256 256 256 Output]	iv	v	vi
[Input 512 512 Output]	vii	$viii$	ix
[Input 512 512 512 Output]	x	xi	xii

For generating convolution data, we choose the following structure

$$conv[128] - relu - batchnorm - conv[256] - relu -$$
$$batchnorm - pool - conv[512] - relu - batchnorm -$$
$$conv[256] - relu - batchnorm - conv[64] - relu -$$
$$batchnorm - pool - (extract\ data) - dense[512] -$$
$$batchnorm - dropout - dense[128] - batchnorm -$$
$$dropout - dense[output]$$

with kernel size 3×3(strides 1) and pool size 2×2(strides 2), then train this CNN with batch size 64, learning rate 0.001 and optimizer RMSProp for 5 epochs. After that, extract new data at location mentioned in above structure by feeding the data to the trained network.

Appendix

C

Extraction of Mechanical Properties of Materials through Deep Learning from Instrumented Indentation

C.1 Nomenclature for forward and inverse problems in instrumented sharp indentation

Fig. 4.1A (left side) is a schematic diagram showing a typical stress-strain response of a power-law strain-hardening material which, to a good approximation, can be used for many engineering metallic materials with isotropic properties (comprising nearly equi-axed grains). The elastic behavior follows Hook's law, whereas the plastic response typically can be represented by the Von Mises yield criterion and power-law strain hardening. True stress σ and true strain ε are related as:

$$\sigma = \begin{cases} E\varepsilon, & \text{for } \sigma \leq \sigma_y \\ R\varepsilon^n, & \text{for } \sigma \geq \sigma_y \end{cases} \tag{C.1}$$

where E is the Young's modulus, R a strength coefficient, n the strain hardening exponent and σ_y the initial yield stress at zero offset strain. In the plastic deformation region, true strain can be further decomposed to strain at yield and true plastic strain: $\varepsilon = \varepsilon_y + \varepsilon_p$. At the yielding point, the following condition must hold:

$$\sigma_y = E\varepsilon_y = R\varepsilon^n. \tag{C.2}$$

Thus, when $\sigma > \sigma_y$, eqs. (C.1)-(C.2) combine to become

$$\sigma = \sigma_y \left(1 + \frac{E}{\sigma_y}\varepsilon_p\right)^n. \tag{C.3}$$

When an indenter is in contact with a substrate material of different properties, the reduced modulus, E^*, is often used to simplify the problem. E^* is defined as

$$E^* = \left[\frac{1-\nu^2}{E} + \frac{1-\nu_i^2}{E_i} \right]^{-1} \tag{C.4}$$

where E is Young's modulus of the substrate material (the material being indented), and ν is its Poisson's ratio; E_i is Young's modulus of the indenter, and ν_i is its Poisson's ratio.

Fig. 4.1A (right side) also schematically shows the typical indentation load versus displacement, i.e., the $P-h$ response, of an elastoplastic material subjected to sharp indentation. The theoretical loading response for a sharp indenter tip is governed by Kick's Law [103],

$$P = Ch^2, \tag{C.5}$$

where C is the loading curvature. (This law is expected to hold when the depth of penetration of the indenter tip into the substrate is at least several times greater than the tip radius of the indenter, and the indenter contact diameter is at least on the order of ten times the depth of penetration into the substrate. In addition, note that for continuum analyses to hold, all the characteristic dimensions of indentation discussed here must be at least several times larger than the average structural dimension of the material with its characteristic microstructural features, such as particle size, grain size, etc.)

At the maximum depth of penetration h_m, the corresponding indentation load P_m makes a projected contact area of A_m. The average contact pressure is thus defined as $p_{ave} = \frac{P_m}{A_m}$, commonly referred to as the hardness of the indented material. Upon unloading, the initial unloading slope is defined as $\frac{dP}{dh}|_{h_m}$. Upon

complete unloading, the residual depth from the indentation impression is h_r. The area under the loading portion is defined as the total work W_t; the area under the unloading portion is defined as the recovered elastic work W_e; and the area enclosed by the loading and unloading portions is defined as the residual plastic work $W_p = W_t - W_e$. See Fig. 4.1 for graphical representations of these parameters and more details in ref. [50].

A representative plastic strain ε_r can be defined as a strain level, which allows for the construction of a dimensionless description of indentation loading response for a particular geometry of the sharp indenter tip, independent of the strain hardening exponent n. A comprehensive framework using dimensional analysis to extract closed form universal functions for the representative plastic strain has been developed [50, 46] based on brute force finite element simulations. Values of ε_r for different indenter geometries are also available in the literature [50, 46]. Note that particular values of ε_r depend strongly on how it is defined [50].

We constructed and used universal dimensionless functions for single sharp indentation [50] and dual/multiple indentation with two indenter tip geometries [46, 28, 31, 220] to formulate forward and inverse algorithms. Issues of accuracy, sensitivity and uniqueness have been discussed in [50, 121]. In brief, the forward algorithms are robust with low sensitivity while the inverse algorithms are more sensitive to small experimental errors in extracting elastoplastic properties. We also note that the uniqueness of the inverse solution is not always guaranteed for certain parameter ranges, especially when solving the single indentation inverse problem [50, 43]. More details can be found in refs. [50, 46, 74].

C.2 Inverse analysis using the multi-fidelity Gaussian process

We test the same problem in Section "Inverse Analysis Using MFNNs" (Approach 1) using the linear multi-fidelity Gaussian process regression (MFGPR), which is a widely used multi-fidelity method despite being only suitable for problems where the correlation between low-fidelity and high-fidelity outputs is almost linear. Because the computational cost of MFGPR scales as N^3 (with N being the number of training data sets), we only use 1,000 low-fidelity data points for comparison. When 70 high-fidelity points are used, the errors obtained by using MFGPR for E^* and σ_y are $23 \pm 19\%$ and $38 \pm 16\%$, respectively. These values are much larger than the errors seen in the present study using MFNN. MFGPR thus fails to do well in solving the inverse indentation problem because the correlation between low- and high-fidelity data sets is no longer linear (see Fig. C.3).

C.3 Tip-radius-effect correction and tip radius estimation

In ref. [45], a simplified tip-radius-effect correction method was proposed and verified to be a good approximation for obtaining the corrected load (P) versus indenter displacement (h) response with a known tip radius a_{tip}. Briefly, the tip-radius effect towards the load-displacement curve can be approximated by a corresponding shift, h_b, to the $P - h$ curve. In the present study, we first estimate the values of h_b corresponding to different tip radii a_{tip} in 100 nm intervals. The corrected $P - h$ curves shifted by different h_b values are then compared with FEM

generated theoretical curves. The best match of $(C, \frac{dP}{dh}, \frac{W_p}{W_t})$ parameter set gives the chosen h_b value used for tip-radius-effect correction, which corresponds to an estimated tip radius of 0.6 μm.

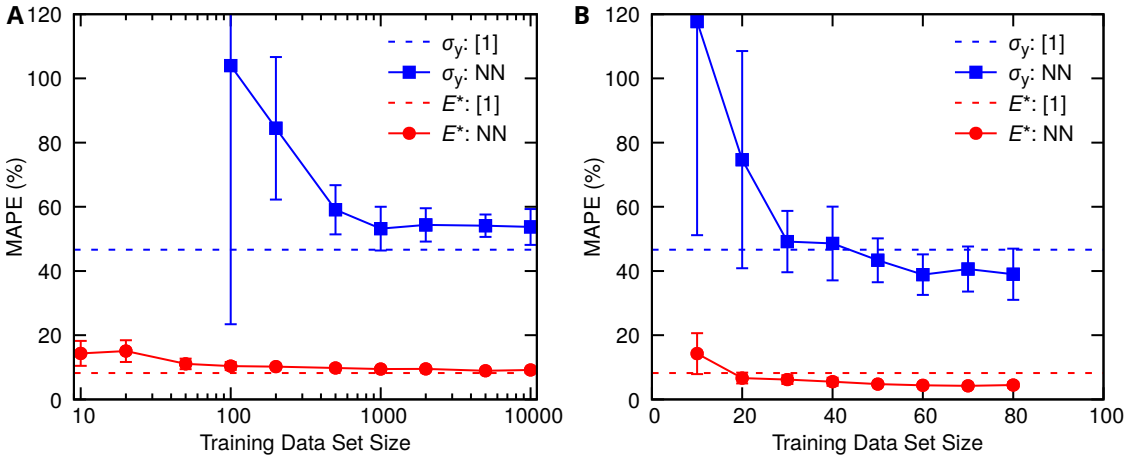


Figure C.1: Single-fidelity NNs for inverse indentation problems. (**A**) Results obtained using training data generated from previously established dimensionless fitting functions in ref. [50]. The dotted lines show the average error of directly applying the equations for E^* and σ_y. The solid red and blue lines show the NN errors of E^* and σ_y, respectively, versus the different training data set size. (**B**) Results obtained using 2D axisymmetric finite element conical indentation data for training. The dotted lines show the average error of directly applying the previous established dimensionless fitting functions in [50]. The solid red and blue lines show the NN errors of E^* and σ_y, respectively, versus the different training data set size. All data in (A) and (B) are assuming a conical indenter with a half-included tip angle of 70.3°.

C.4 Sensitivity of the inverse indentation problem to extracting plastic properties

The inverse indentation problem of extracting plastic properties is inherently sensitive to small experimental errors. Regarding the dependence of yield strength and strain-hardening exponent to such experimental errors, Fig. C.5 shows results from using the equations in ref. [50] (Fig. C.5A, B and C) and using NNs trained using 2D+3D FEM data for estimating errors (Fig. C.5D, E and F) with respect to σ_y, σ_y/E and n, respectively. Note that there is, in general, not a clear dependence

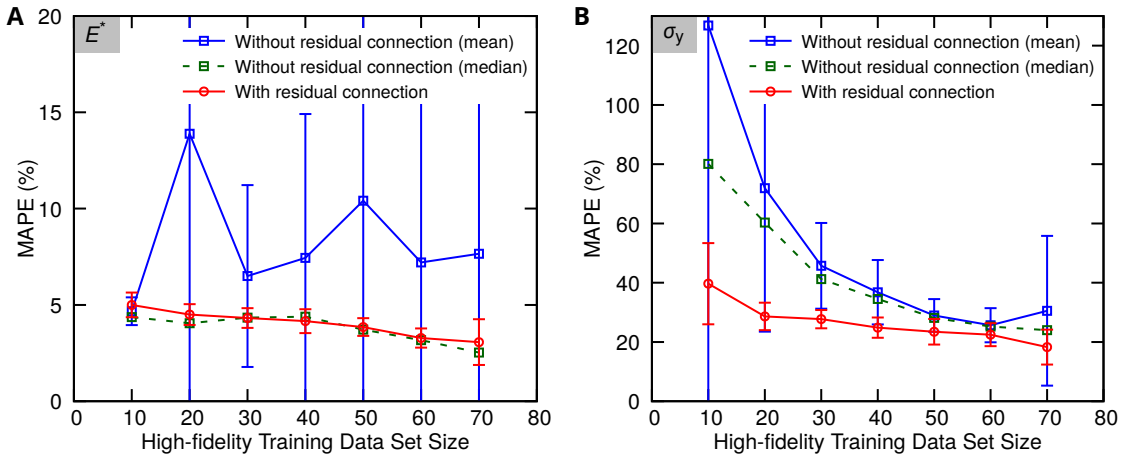


Figure C.2: Comparison between the proposed multi-fidelity neural network (MFNN) based on the residual connection with the original MFNN. The blue and green lines represent the mean and median of errors of the original MFNN for (**A**) E^* and (**B**) σ_y, and the red line represents the mean MAPE of the residual MFNN. By adding the residual connection, the training process becomes more stable, and the error is also significantly reduced.

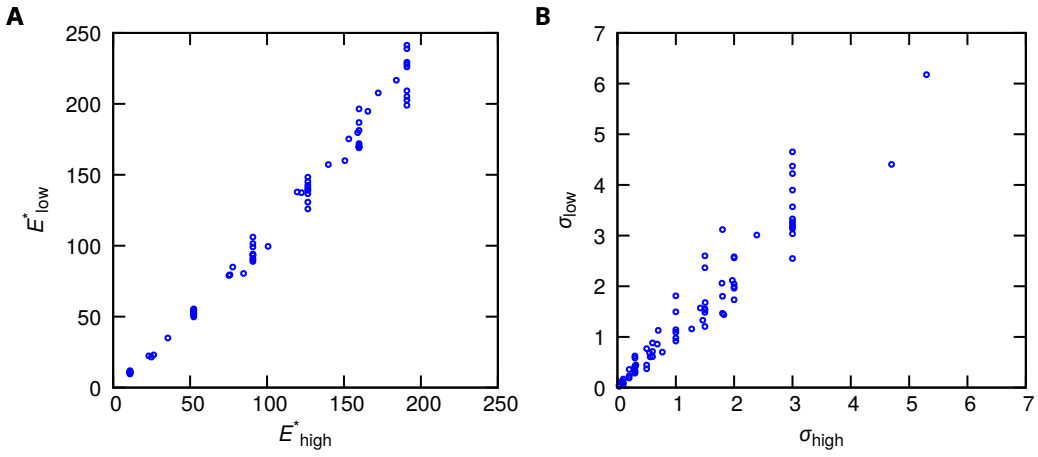


Figure C.3: The non-linear correlation between low-fidelity formulas and high-fidelity FEM data for (**A**) E^* and (**B**) σ_y. E^*_{low} and E^*_{high} of each point represent the values of E^* obtained from fitting functions and from the FEM data for the same $(C, \frac{dP}{dh}, \frac{W_p}{W_t})$ in (A). Same for σ_y in (B).

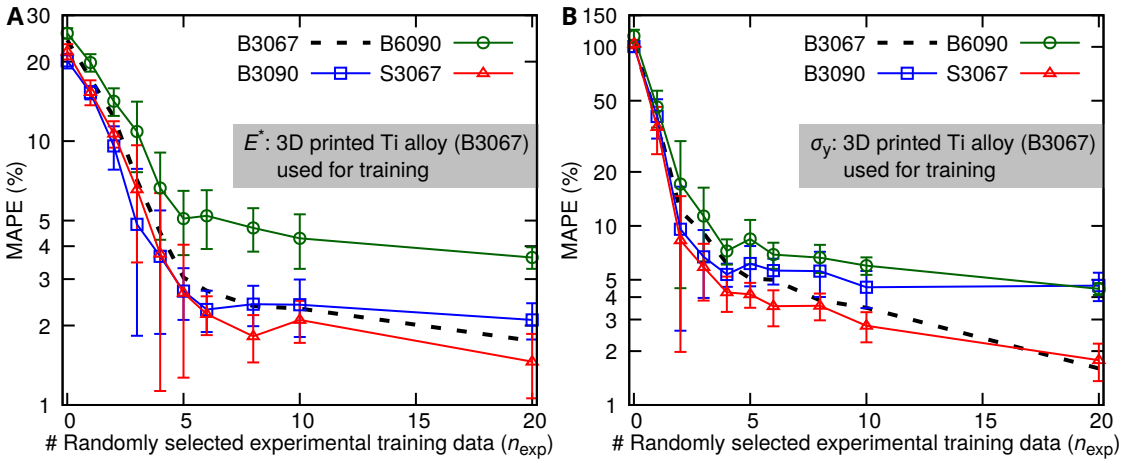


Figure C.4: Inverse analysis of four 3D printed Ti-6Al-4V alloys. (**A**) E^* and (**B**) σ_y for 3D printed Ti-6Al-4V alloys (B3067, B3090, B6090, and S3067) versus the number of randomly selected experimental training data from B3067 indentation experiments. All experimental data used are from the uncorrected raw indentation data.

of error on either σ_y/E or n in all cases. For results using the universal equations in ref. [50], somewhat larger errors are found when σ_y/E is less than 0.1 or close to 0.4, and when n is close to zero or between 0.15 to 0.25. The 2D+3D FEM trained NNs show smaller average errors and less scatter in terms of errors across the entire parameter space than the cases using the equations from ref. [50].

In addition, the proposed method in the present study has been shown to be very effective in learning and reducing systematic errors in indentation experiments, when a small number of high-fidelity experimental data points is added for training. These are shown clearly in the presented results for modulus, yield strength and strain-hardening behavior. This is due to that fact that, with a nominally homogeneous material, instrumented indentation experiments are known to produce highly repeatable force vs penetration depth curves when using the same indenter instrument and the same setup. By employing a reliable method that can effectively learn and correct the involved systematic errors, we can use the method to calibrate the indenter and obtain reliable inverse analyses results.

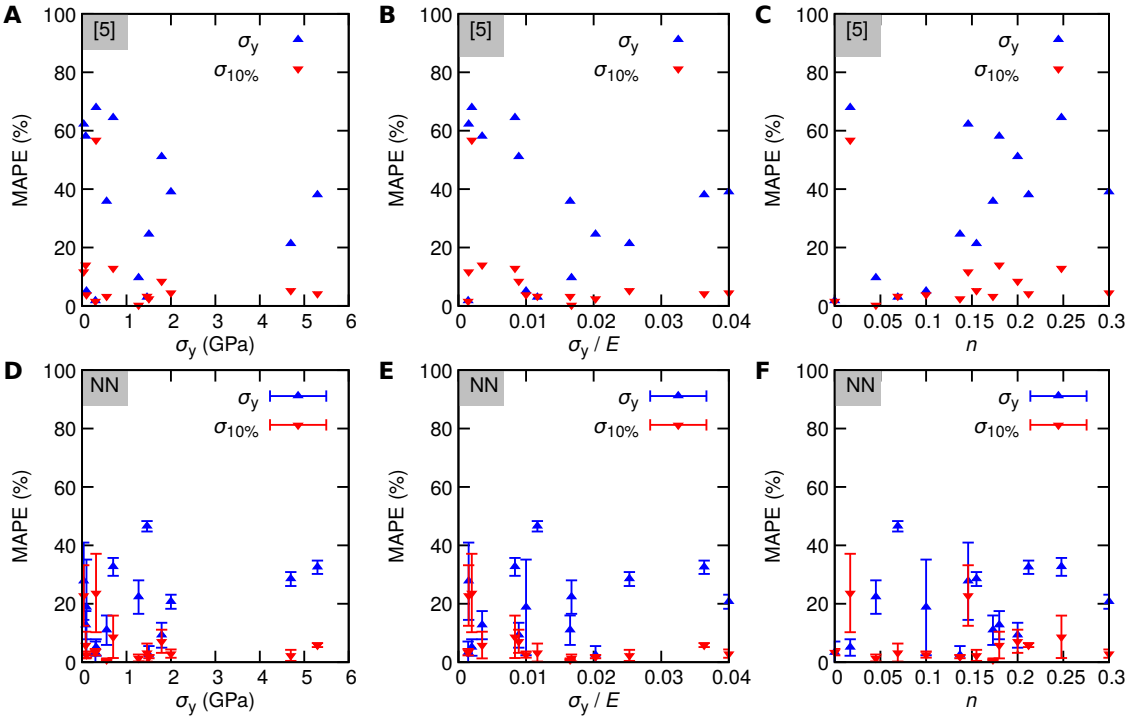


Figure C.5: Mean average percentage error of σ_y and $\sigma_{10\%}$ as a function of σ_y, σ_y/E and n using (**A**, **B** and **C**) fitting functions in ref. [50] and (**D**, **E** and **F**) multi-fidelity NNs trained by 2D and 3D FEM simulations for solving the inverse indentation problem. Results of multi-fidelity NNs are trained by integrating 2D axisymmetric FEM results (low fidelity) together with 3D FEM simulation data (high fidelity). There are totally 14 data points of 3D FEM simulation; 13 randomly selected data points are used in training, and the remaining one is used for testing; all 14 possible such selections are studied and summarized using error bars in the figure.

Table C.1: Input parameters for neural network training for different cases. The output for all cases is either E^* or σ_y.

Case	Inputs
Single fidelity 1 indenter	$C, dP/dh, W_p/W_t$ in 70.3°
Single fidelity 2 indenters	$(C, dP/dh, W_p/W_t)$ in 70.3°, and C in 60°
Single fidelity 4 indenters	$(C, dP/dh, W_p/W_t)$ in 70.3°, C in 50°, C in 60°, and C in 80°
Multi fidelity	$C, dP/dh, W_p/W_t$

Table C.2: The datasets and the sizes of the datasets used in this study.

Dataset	Size
2D FEM in 50°	76
2D FEM in 60°	100
2D FEM in 70.3°	100, including 3 data points with $n > 0.3$ and $\sigma_y/E^* \geq 0.03$
2D FEM in 80°	76
3D Berkovich FEM	14
Al6061-T6511	6
Al7075-T651	6
B3067	144
B3090	144
B6067	144
B6090	144
S3067	144
S6067	144

Table C.3: Indentation characteristics extracted from Berkovich indentation curves of two aluminum alloys Al6061-T6511 and Al7075-T651, with a maximum indentation load of 3 N for each indentation experiment.

Material	C (GPa)	dP/dh (kN/m)	W_p/W_t
Al6061-T6511	27.4 ± 0.4	5086 ± 138	0.896 ± 0.008
Al7075-T651	42.7 ± 1.3	4005 ± 52	0.835 ± 0.003

Table C.4: Indentation characteristics extracted (**A**) directly from raw indentation curves of six 3D printed Ti-6Al-4V alloys (B3067, B3090, B6067, B6090, S3067 and S6067) and (**B**) from indentation curves corrected with a indenter tip radius of 0.6 µm for two 3D printed Ti-6Al-4V alloys (B3067 and B3090), using the same experimental setup with a maximum indentation load of 9 mN for each indentation experiment. Here, B and S refer to the build and transverse orientations, the first two digits refer to the thickness (in µm) of each layer (employed in the layer-by-layer 3D printing process), and the last two digits indicate the scan rotation between successive layers. The standard deviations shown in the table are from 144 indentation curves. The corresponding uniaxial stress-strain curves can be found in ref. [116].

A

Material	C (GPa)	dP/dh (kN/m)	W_p/W_t
B3067	137.7 ± 9.4	202.7 ± 4.2	0.73 ± 0.01
B3090	137.4 ± 9.0	195.9 ± 3.9	0.72 ± 0.01
B6067	141.1 ± 9.8	199.5 ± 3.8	0.72 ± 0.01
B6090	143.0 ± 19.2	197.5 ± 4.3	0.71 ± 0.01
S3067	135.9 ± 8.5	201.4 ± 3.9	0.72 ± 0.01
S6067	135.1 ± 10.5	199.6 ± 3.8	0.73 ± 0.01

B

Material	C (GPa)	dP/dh (kN/m)	W_p/W_t
B3067	96.3 ± 5.4	202.7 ± 4.2	0.73 ± 0.01
B3090	96.2 ± 5.3	195.9 ± 3.9	0.72 ± 0.01

Appendix

D

DeepONet: Learning Nonlinear Operators based on the Universal Approximation Theorem of Operators

D.1 Neural networks to approximate nonlinear operators

We list in Table D.1 the main symbols and notations that are used throughout this chapter.

Table D.1: Notation.

X	a Banach space with norm $\|\cdot\|_X$
$\mathbb{R}^d$	Euclidean space of dimension d
K	a compact set in a Banach space
$C(K)$	Banach space of all continuous functions defined on K with norm $\|f\|_{C(K)} = \max_{x\in K} \|f(x)\|$
V	a compact set in $C(K)$
$u(x)$	an input function/signal
$s(x)$	an output function/signal
f	a function or functional
G	an operator
(TW)	all the Tauber-Wiener functions
σ	an activation function
$\{x_1, x_2, \ldots, x_m\}$	m sensor points
n, p	neural network size hyperparameters in Theorems D.1.4 and D.1.5 below
N	number of basis functions
M	value of some upper bound

Let $C(K)$ denote the Banach space of all continuous functions defined on a compact set $K \subset X$ with sup-norm $\|f\|_{C(K)} = \max_{x\in K} |f(x)|$, where X is a Banach space. We first review the definition and sufficient condition of Tauber-Wiener (TW) functions [37], and the definition of continuous operator.

Definition D.1.1 (TW). *If a function $\sigma : \mathbb{R} \to \mathbb{R}$ (continuous or discontinuous) satisfies that all the linear combinations $\sum_{i=1}^{N} c_i \sigma(\lambda_i x + \theta_i)$, $\lambda_i \in \mathbb{R}$, $\theta_i \in \mathbb{R}$, $c_i \in \mathbb{R}$, $i = 1, 2, \ldots, N$, are dense in every $C([a, b])$, then σ is called a Tauber-Wiener (TW) function.*

Theorem D.1.2 (Sufficient condition for TW). *Suppose that σ is a continuous function, and $\sigma \in S'(\mathbb{R})$ (tempered distributions), then $\sigma \in (TW)$, if and only if σ is not a polynomial.*

It is easy to verify that all the activation functions we used nowadays, such as sigmoid, tanh and ReLU, are TW functions.

Definition D.1.3 (Continuity). *Let G be an operator between topological spaces X and Y. We call G continuous if for every $\epsilon > 0$, there exists a constant $\delta > 0$ such that*

$$\|G(x) - G(y)\|_Y < \epsilon$$

for all $x, y \in X$ satisfying $\|x - y\|_X < \delta$.

We recall the following two main theorems of approximating nonlinear continuous functionals and operators due to Chen & Chen [37].

Theorem D.1.4 (Universal Approximation Theorem for Functional). *Suppose that $\sigma \in (TW)$, X is a Banach Space, $K \subset X$ is a compact set, V is a compact set in $C(K)$, f is a continuous functional defined on V, then for any $\epsilon > 0$, there are a positive integer n, m points $x_1, \ldots, x_m \in K$, and real constants c_i, θ_i, ξ_{ij}, $i = 1, \ldots, n$, $j = 1, \ldots, m$, such that*

$$\left| f(u) - \sum_{i=1}^{n} c_i \sigma \left(\sum_{j=1}^{m} \xi_{ij} u(x_j) + \theta_i \right) \right| < \epsilon$$

holds for all $u \in V$.

Theorem D.1.5 (**Universal Approximation Theorem for Operator**). *Suppose that* $\sigma \in (TW)$, X *is a Banach Space,* $K_1 \subset X$, $K_2 \subset \mathbb{R}^d$ *are two compact sets in* X *and* $\mathbb{R}^d$, *respectively,* V *is a compact set in* $C(K_1)$, G *is a nonlinear continuous operator, which maps* V *into* $C(K_2)$, *then for any* $\epsilon > 0$, *there are positive integers* n, p, m, *constants* $c_i^k, \xi_{ij}^k, \theta_i^k, \zeta_k \in \mathbb{R}$, $w_k \in \mathbb{R}^d$, $x_j \in K_1$, $i = 1, \ldots, n$, $k = 1, \ldots, p$, $j = 1, \ldots, m$, *such that*

$$\left| G(u)(y) - \sum_{k=1}^{p} \sum_{i=1}^{n} c_i^k \sigma \left(\sum_{j=1}^{m} \xi_{ij}^k u(x_j) + \theta_i^k \right) \sigma(w_k \cdot y + \zeta_k) \right| < \epsilon$$

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

The necessary and sufficient conditions of the compactness are given by the following Arzelà-Ascoli Theorem.

Definition D.1.6 (**Uniform Boundedness**). *Let* V *be a family of real-valued functions on the set* K. *We call* V *uniformly bounded if there exists a constant* M *such that*

$$|f(x)| \leq M$$

for all $f \in V$ *and all* $x \in K$.

Definition D.1.7 (**Equicontinuity**). *Let* V *be a family of real-valued functions on the set* K. *We call* V *equicontinuous if for every* $\epsilon > 0$, *there exists a* $\delta > 0$ *such that*

$$|f(x) - f(y)| < \epsilon$$

for all $f \in V$ *and all* $x, y \in K$ *satisfying* $|x - y| < \delta$.

Theorem D.1.8 (**Arzelà-Ascoli Theorem**). *Let* X *be a Banach space, and* $K \subset X$ *be a compact set. A subset* V *of* $C(K)$ *is pre-compact (has compact closure) if and only if* V *is uniformly bounded and equicontinuous.*

D.2 Data generation

Here, we list the details of function spaces V and reference solutions.

Gaussian random fields (GRFs). In most examples, we use the mean-zero GRFs for the input function $u(x)$:

$$u(x) \sim \mathcal{GP}(0, k_l(x_1, x_2)),$$

where the covariance kernel $k_l(x_1, x_2) = \exp(-\|x_1 - x_2\|^2/2l^2)$ is the Gaussian kernel with a length-scale parameter $l > 0$. The length-scale l determines the smoothness of the sampled function, and larger l leads to smoother u.

Orthogonal polynomials. Let $M > 0$ and P_i be i-th orthogonal polynomial basis. We define the N-dimensional orthogonal polynomial space as

$$V_{\text{Org}} = \left\{ \sum_{i=0}^{N-1} a_i P_i(x) : |a_i| \leq M \right\}.$$

We generate the samples of input function $u(\cdot)$ from the space V_{Org} by randomly sampling the expansion coefficient a_i from $[-M, M]$.

In the chapter, we consider four different bases $P_i(\cdot)$:

1. the Chebyshev polynomial of the first kind $T_i(x)$;
2. the Legendre polynomial $L_i(x)$;
3. the poly-fractonomial polynomial $(1 + x)^{0.5} J_i^{-0.5,0.5}(x)$, where $J_i^{\alpha,\beta}(x)$ is the

Jacobi polynomial of degree i;

4. the Zernike polynomial $Z_i(r, \theta)$, where $Z_i(r, \theta)$ is the single indexing Zernike polynomial whose index i is linked to the indices of double-indexing one $Z_n^m(r, \theta)$ by $i = \frac{n(n+2)+m}{2}$.

Reference solutions. The reference solutions of all ODE systems are obtained by Runge-Kutta (4, 5), and the reference solutions of all deterministic PDEs are obtained by a second-order finite difference method.

The exact solution of the stochastic ODE is

$$y = y_0 e^{K(t)},$$

where $K(t) = \int_0^t k(s)\, ds$ is again Gaussian process with mean zero.

The exact solution of the stochastic PDE is

$$u(x) = C(\omega) \int_0^x e^{-b(y;\omega)}\, dy - \int_0^x \int_0^y e^{-b(y;\omega)} f(z)\, dz\, dy,$$

where

$$C(\omega) = \left(\int_0^1 e^{-b(y;\omega)}\, dy\right)^{-1} \int_0^1 \int_0^y e^{-b(y;\omega)} f(z)\, dz\, dy.$$

Numerical solutions are obtained by approximating $k(t;\omega)$ or $b(x;\omega)$ with their truncated Karhunen-Loeve expansions.

Remark. We note that one data point is a triplet $(u, y, G(u)(y))$, and thus one specific input u may appear in multiple data points with different values of y. For example, a dataset of size 10000 may only be generated from 100 u trajectories,

and each evaluates $G(u)(y)$ for 100 y locations.

D.3 Gaussian random field with the Gaussian kernel

Suppose that $X(t) \sim \mathcal{GP}(0, \exp(-\frac{|x-y|^2}{l^2}))$. Then, by the Wiener-Khinchin Theorem, we have

$$X(t) = \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} (l)^{\frac{1}{2}} \cos(\omega t) \exp(-\frac{l^2\omega^2}{8}) dW(\omega) \\ - \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} (l)^{\frac{1}{2}} \sin(\omega t) \exp(-\frac{l^2\omega^2}{8}) dB(\omega),$$

where W and B are independent standard Brownian motions [238]. Apply the change of variable $\lambda = l\omega$ and write $X(t)$ as

$$X(t) = \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} \cos(\frac{\lambda}{l}t) \exp(-\frac{\lambda^2}{8}) dW(\lambda) - \sqrt{2}(\pi)^{\frac{1}{4}} \int_{\mathbb{R}^+} \sin(\frac{\lambda}{l}t) \exp(-\frac{\lambda^2}{8}) dB(\lambda).$$

Applying a linear interpolation Π_1 on the interval $[t_i, t_{i+1}]$, then

$$\begin{aligned} \mathbb{E}[(X(t) - \Pi_1 X(t))^2] &= 2(\pi)^{\frac{1}{2}} \int_{\mathbb{R}^+} ((I - \Pi_1)\cos(\frac{\lambda}{l}t))^2 \exp(-\frac{\lambda^2}{4}) d\lambda \\ &\quad + 2(\pi)^{\frac{1}{2}} \int_{\mathbb{R}^+} ((I - \Pi_1)\sin(\frac{\lambda}{l}t))^2 \exp(-\frac{\lambda^2}{4}) d\lambda \\ &\leq (\pi)^{\frac{1}{2}} (t_{i+1} - t_i)^4 \int_{\mathbb{R}^+} (\frac{\lambda}{l})^4 \exp(-\frac{\lambda^2}{4}) d\lambda \\ &= 24\pi \frac{(t_{i+1} - t_i)^4}{l^4}, \end{aligned}$$

where we recalled the error estimate of the linear interpolation on $[a, b]$ (by Taylor's expansion)

$$|(I - \Pi_1)g(t)| = \frac{1}{2}(b-a)^2 |g''(\xi)|,$$

where ξ lies in between a and b. Then by the fact that $X(t) - \Pi_1 X(t)$ is Gaussian, we have

$$|X(t) - \Pi_1 X(t)| \leq C \frac{(t_{i+1} - t_i)^2}{l^2}, \quad \epsilon > 0,\ t \in [t_i, t_{i+1}],$$

where C is an absolute value of a Gaussian random variable with mean zero and variance $\sqrt{24\pi}$. In conclusion, taking a piecewise linear interpolation of $X(t)$ with m points will lead to convergence with order $O(\frac{1}{m^2 l^2})$.

D.4 Number of sensors for identifying nonlinear dynamical systems

This part is the proof of Theorem 6.2.2. All the concepts and notations involved here are defined in the paragraph of Theorem 6.2.2.

Lemma D.4.1. *$W \coloneqq \bigcup_{i=1}^{\infty} W_i$ is compact.*

Proof. At first, we prove that W is pre-compact. For any $\varepsilon > 0$, by (6.3), there exists an m_0 such that

$$\|u - \mathcal{L}_m(u)\|_C < \frac{\varepsilon}{4}, \quad \forall u \in V, \forall m > m_0.$$

Since W_{m_0} is a compact set subject to equicontinuity, there exists a $\delta > 0$ such that

$$|x - y| < \delta \Rightarrow |u(x) - u(y)| < \frac{\varepsilon}{2}, \quad \forall u \in W_{m_0}, \forall x, y \in [a, b].$$

Now for all $u \in W$ and all $x, y \in [a, b], |x - y| < \delta$, if $u \in W_{m_0}$, naturally we have $|u(x) - u(y)| < \frac{\varepsilon}{2} < \varepsilon$, otherwise, $u \in \bigcup_{i=m_0+1}^{\infty} U_i$. Suppose that $u = \mathcal{L}_m(v), m > m_0, v \in V$, then there holds

$$\begin{aligned}
|u(x) - u(y)| =& |u(x) - v(x) + v(x) - v(y) + v(y) - u(y)| \\
\leq& |\mathcal{L}_m(v)(x) - v(x)| + |v(x) - v(y)| + |\mathcal{L}_m(v)(y) - v(y)| \\
\leq& 2\|\mathcal{L}_m(v) - v\|_C + |v(x) - v(y)| \\
<& 2 \cdot \frac{\varepsilon}{4} + \frac{\varepsilon}{2} = \varepsilon,
\end{aligned}$$

which shows the quicontinuity of W. In addition, it is obvious that W is uniformly bounded, so that we know W is pre-compact by applying the Arzelà-Ascoli Theorem.

Next we show that W is closed. Let $\{w_i\}_{i=1}^{\infty} \subset W$ be a sequence which converges to a $w_0 \in C[a,b]$. If there exists an m such that $\{w_i\} \subset W_m$, then $w_0 \in W_m \subset W$. Otherwise, there is a subsequence $\{\mathcal{L}_{i_n}(v_{i_n})\}$ of $\{w_i\}$ such that $v_{i_n} \in V$ and $i_n \to \infty$ as $n \to \infty$. Then we have

$$\begin{aligned}\|v_{i_n} - w_0\|_C =& \|v_{i_n} - \mathcal{L}_{i_n}(v_{i_n}) + \mathcal{L}_{i_n}(v_{i_n}) - w_0\|_C \\ \leq& \|v_{i_n} - \mathcal{L}_{i_n}(v_{i_n})\|_C + \|\mathcal{L}_{i_n}(v_{i_n}) - w_0\|_C \\ \leq& \kappa(i_n, V) + \|\mathcal{L}_{i_n}(v_{i_n}) - w_0\|_C,\end{aligned}$$

which implies that $w_0 = \lim_{n\to\infty} v_{i_n} \in V \subset W$. □

Next we show the proof of Theorem 6.2.2.

Proof. For $u \in V$ and $u_m \in U_m$, according to the bound (6.3) and the Lipschitz conditions of the right-hand side of the ODE, we can derive that the operator is Lipschitz continuous.

$$\begin{aligned}\|G(u)(d) - G(u_m)(d)\|_2 &\leq c\int_a^d \|G(u)(t) - G(u_m)(t)\|_2 dt + c\int_a^d |u(t) - u_m(t)| dt \\ &\leq c\int_a^d \|G(u)(t) - G(u_m)(t)\|_2 dt + c(b-a)\kappa(m, V).\end{aligned}$$

Here $\kappa(m, V)$ is defined in (6.3). By Grönwall's inequality, we have

$$\|G(u)(d) - G(u_m)(d))\|_2 \leq c(b-a)\kappa(m, V)e^{c(b-a)}.$$

Define $S_m = \{(u(x_0), u(x_1), \cdots, u(x_m)) \in \mathbb{R}^{m+1} | u \in V\}$. Then S_m is a compact set in $\mathbb{R}^{m+1}$ as it is the image of a continuous map on the compact set V. As there is a bijective map between S_m and U_m, we may define a vector-valued function φ on

S_m by

$$\varphi(u(x_0), u(x_1), \cdots, u(x_m)) = G(u_m)(d).$$

Because G is a continuous operator over V, φ is a continuous function over S_m. For any $\varepsilon > 0$, make m large enough so that $c(b-a)\kappa(m, V)e^{c(b-a)} < \varepsilon$. By the universal approximation theorem of neural network for high-dimensional functions, there exist $\mathcal{W}_1 \in \mathbb{R}^{n\times(m+1)}, b_1 \in \mathbb{R}^{m+1}, \mathcal{W}_2 \in \mathbb{R}^{K\times n}, b_2 \in \mathbb{R}^K$, such that

$$\begin{aligned}\|\varphi(u(x_0), \cdots, u(x_m)) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 \\ < \varepsilon - c(b-a)\kappa(m, V)e^{c(b-a)}\end{aligned}$$

holds for all $(u(x_0), \cdots, u(x_m)) \in S_m$. Hence, we have

$$\begin{aligned}&\|(Gu)(d) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 \\ \leq &\|(Gu)(d) - (Gu_m)(d)\|_2 \\ &+ \|(Gu_m)(d) - (\mathcal{W}_2 \cdot \sigma(\mathcal{W}_1 \cdot [u(x_0) \quad \cdots \quad u(x_m)]^T + b_1) + b_2)\|_2 \\ < &c(b-a)\kappa(m, V)e^{c(b-a)} + \varepsilon - c(b-a)\kappa(m, V)e^{c(b-a)} = \varepsilon.\end{aligned}$$

In summary, by choosing the value of m so that it makes $c(b-a)\kappa(m, V)e^{c(b-a)}$ less than ε is sufficient to achieve accuracy ε. □

D.5 Parameters

In all problems we use ReLU activation function, except that for fractional differential operators we use the hyperbolic tangent. We use the Adam optimizer [112] with learning rate 0.001, except that for the Legendre transform it is 0.0001. The other parameters and network sizes are listed in Tables D.2 and D.3, unless otherwise stated.

Table D.2: Default parameters for each problem, unless otherwise stated. We note that one data point is a triplet $(u, y, G(u)(y))$, and thus one specific input u may generate multiple data points with different values of y.

Case	Input function space	# Sensors m	# Training	# Test	# Iterations
Antiderivative	GRF ($l = 0.2$)	100	10^4	10^5	5×10^4
Legendre transform	GRF ($l = 0.2$)	200	10^4	2×10^4	2×10^6
Fractional (1D)	–	15	10^6	10^6	3×10^4
Fractional (2D)	Zernike	225	1.125×10^7	1.125×10^7	5×10^3
Nonlinear ODE	GRF ($l = 0.2$)	100	10^4	10^5	10^5
Gravity pendulum	GRF ($l = 0.2$)	100	10^4	10^5	10^5
Diffusion-reaction	GRF ($l = 0.2$)	100	–	10^6	5×10^5
Advection	GRF	100	5×10^4	10^5	5×10^5
Advection-diffusion	GRF ($l = 0.5$)	100	5×10^4	10^5	5×10^5
Stochastic ODE/PDE	–	20	10^6	10^6	5×10^4

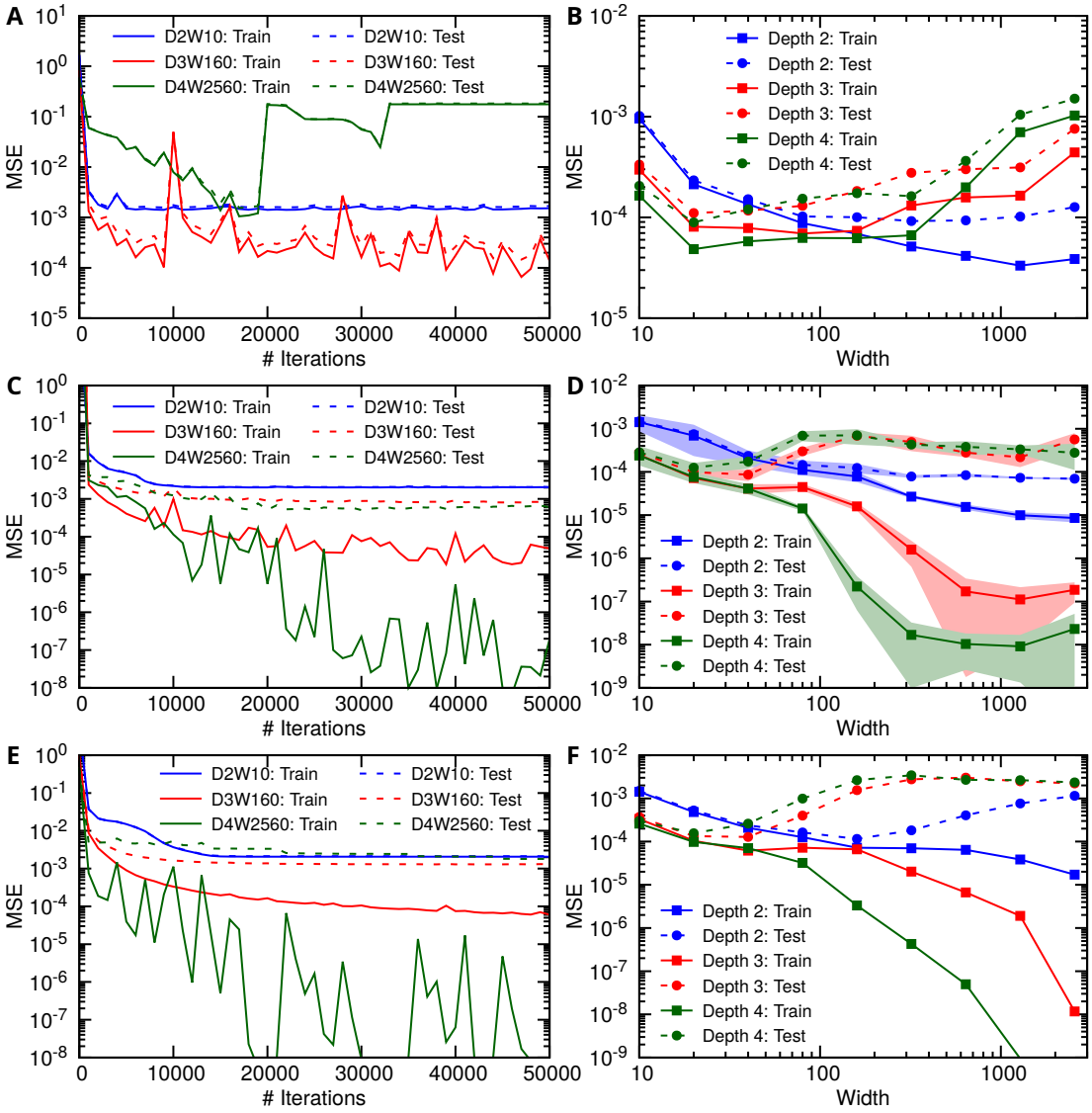


Figure D.1: Errors of FNNs trained to learn the antiderivative operator (linear case). (**A** and **B**) Learning rate 0.01. (**C** and **D**) Learning rate 0.001. (**E** and **F**) Learning rate 0.0001. (A, C, E) The solid and dash lines are the training error and test error during the training process, respectively. The blue, red and green lines represent FNNs of size (depth 2, width 10), (depth 3, width 160), and (depth 4, width 2560), respectively. (B, D, F) The blue, red and green lines represent FNNs of depth 2, 3 and 4, respectively. The shaded regions in (D) are the one-standard-derivation (SD) from 10 runs with different training/test data and network initialization. For clarity, only the SD of learning rate 0.001 is shown (Fig. D). The number of sensors for u is $m = 100$.

Table D.3: DeepONet size for each problem, unless otherwise stated.

Case	Trunk depth	Trunk width	Branch depth	Branch width
Antiderivative	3	40	2	40
Legendre transform	3	100	2	100
Fractional (1D)	3	40	2	40
Fractional (2D)	3	60	2	60
Nonlinear ODE	3	40	2	40
Gravity pendulum	3	40	2	40
Diffusion-reaction	3	100	2	100
Advection	3	100	2	100
Advection-diffusion	3	100	2	100
Stochastic ODE/PDE	3	100	2	100

D.6 Legendre transform

We consider the Legendre transform of a function $f(x)$:

$$\textbf{Problem 7} \qquad \mathcal{J}_n\{f(x)\} = \int_{-1}^{1} P_n(x) f(x) dx, \tag{D.1}$$

where $P_n(x)$ is a Legendre polynomial of degree n. We use DeepONet to learn the Legendre transform $f(x) \mapsto \mathcal{J}_n\{f(x)\}$. Because the Legendre transform decays fast to zero w.r.t. n for a smooth $f(x)$, it is sufficient to consider the first 20 coefficients, i.e., $n \in \{0, 1, 2, \ldots, 19\}$. To generate the dataset, we sampled 500 random $f(x)$ from a GRF with the Gaussian kernel of the length scale $l = 0.2$, and computed the Legendre transform at the first 20 coefficients for each $f(x)$. Thus the dataset size is equal to 10000($= 500 \times 20$). The test MSE is $3.2 \times 10^{-7} \pm 1.9 \times 10^{-7}$, and two prediction examples are shown in Fig. D.2.

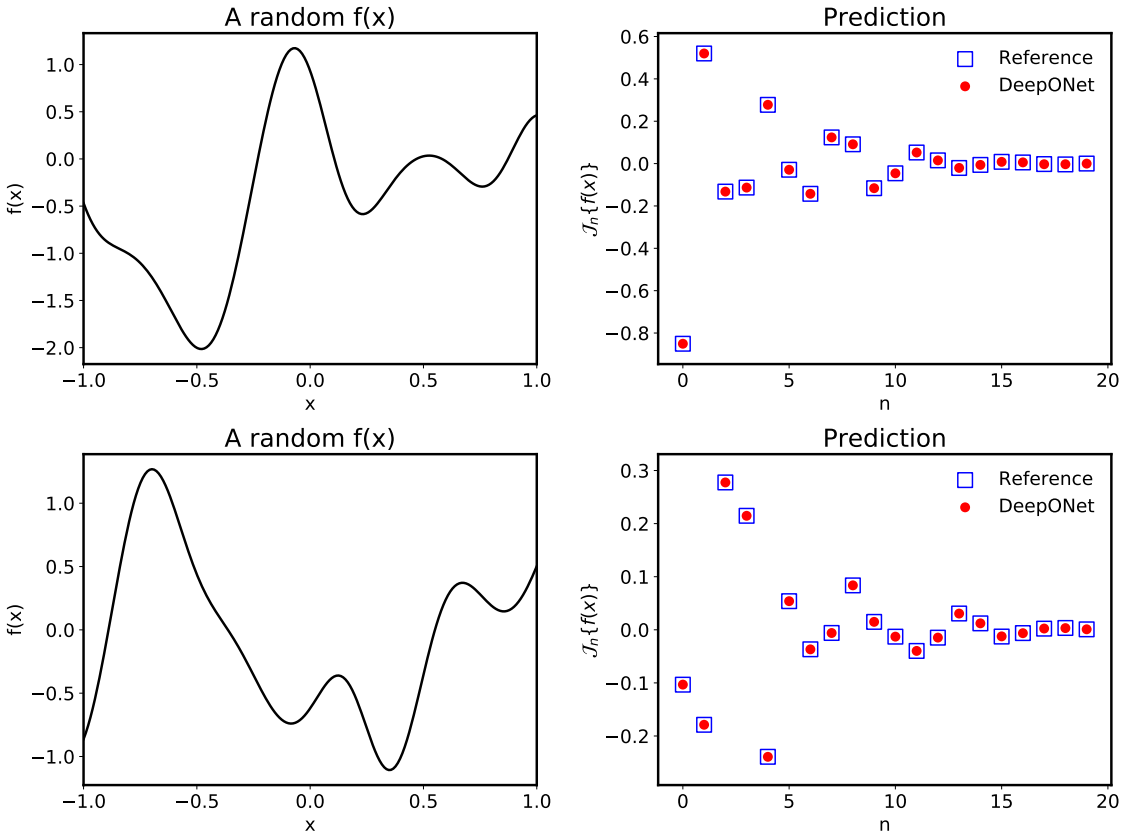


Figure D.2: DeepONet prediction of the Legendre transform for two random $f(x)$.

D.7 Nonlinear ODE system

We consider a nonlinear 1D ODE system

Problem 1.C

$$\frac{ds(x)}{dx} = g(s(x), u(x), x),\ x \in (0, 1],\ \text{where } g(s(x), u(x), x) = -s^2(x) + u(x),$$

with an initial condition $s(0) = 0$. Because $s(x)$ may explode for certain u, we compute the test MSE by removing the 1‰ worst predictions. During the network training, the training MSE and test MSE of both stacked and unstacked DeepONets decrease, but the correlation between training MSE and test MSE of unstacked DeepONets is tighter (Fig. D.3A), i.e., smaller generalization error. This tight correlation between training and test MSE of unstacked DeepONets is also observed across multiple runs with random training dataset and network initialization (Fig. D.3B). Moreover, the test MSE and training MSE of unstacked DeepONets follow almost a linear correlation

$$\text{MSE}_{\text{test}} \approx 10 \times \text{MSE}_{\text{train}} - 10^{-4}.$$

Fig. D.3C shows that unstacked DeepONets have smaller test MSE due to smaller generalization error. DeepONets work even for out-of-distribution predictions, see three examples of the prediction in Fig. D.4.

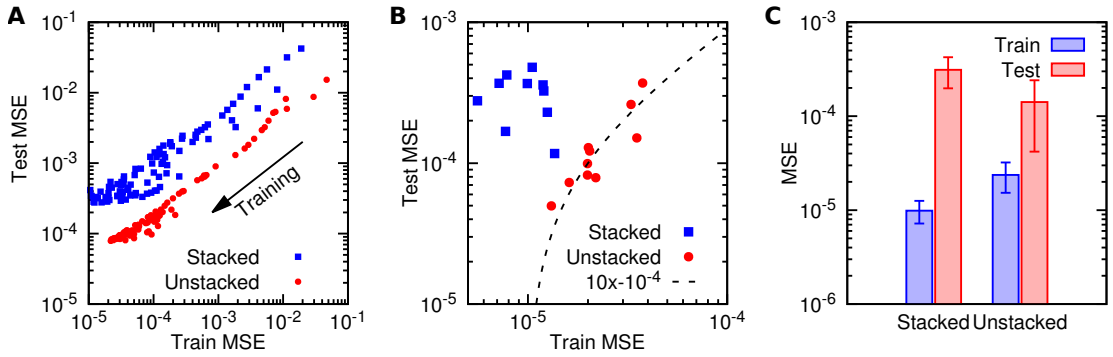


Figure D.3: Nonlinear ODE: unstacked DeepONets have smaller generalization error and smaller test MSE than stacked DeepONets. (**A**) The correlation between the training MSE and the test MSE of one stacked/unstacked DeepONet during the training process. (**B**) The correlation between the final training MSE and test MSE of stacked/unstacked DeepONets in 10 runs with random training dataset and network initialization. The training and test MSE of unstacked DeepONets follow a linear correlation (black dash line). (**C**) The mean and one-standard-derivation of data points in B.

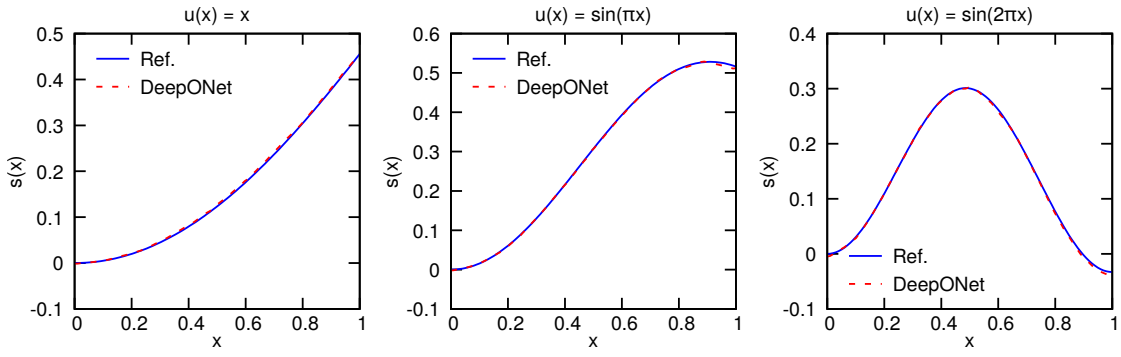


Figure D.4: Nonlinear ODE: predictions of a trained unstacked DeepONet on three out-of-distribution input signals. The blue and red lines represent the reference solution and the prediction of a DeepONet.

D.8 Gravity pendulum with an external force

D.8.1 Number of sensors

The number of sensors required to distinguish two input functions depends on the value of k, prediction time T, and the input function space. For the case with the default parameters $k = 1$, $T = 1$ and $l = 0.2$, when the number of sensors m is small, the error decays exponentially as we increase the number of sensors (Fig. D.5A):

$$\text{MSE} \propto 4.6^{-\#\text{sensors}}.$$

When m is already large, the effect of increasing m is negligible. The transition occurs at ~10 sensors, as indicated by the arrow.

To predict s for a longer time, more sensors are required (Fig. D.5B). For example, predicting until $T = 5$ requires ~25 sensors. If the function u is less smooth corresponding to smaller l, it also requires more sensors (Fig. D.5C). However, the number of sensors is not sensitive to k (Fig. D.5D). Although it is hard to quantify the exact dependency of m on T and l, by fitting the computational results we show that

$$m \propto \sqrt{T} \quad \text{and} \quad m \propto l^{-1}.$$

In Theorem 6.2.2, m should be large enough to make $Te^{cT}\kappa(m, V)$ small. In Section D.3 we show theoretically that $m \propto l^{-1}$ for the GRF function space with the RBF kernel, which is consistent with our computational result here. Te^{cT} in the bound is loose compared to the computational results.

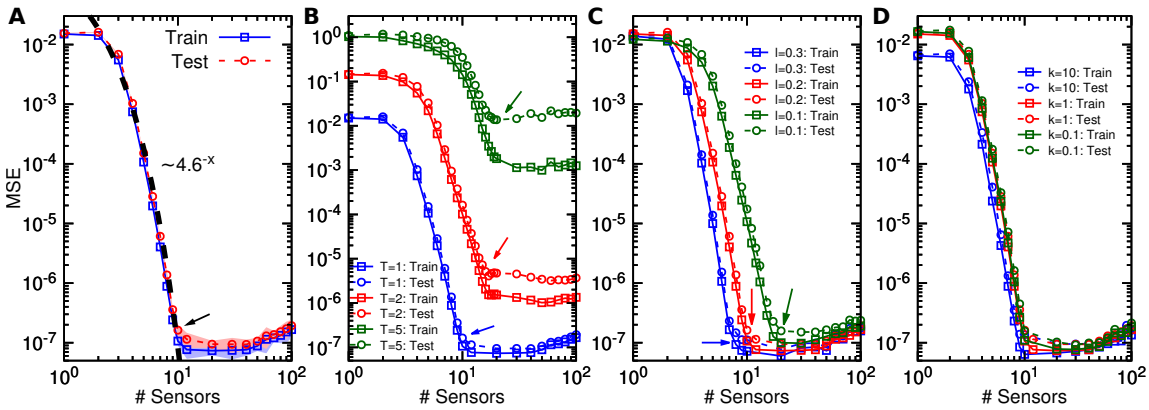


Figure D.5: Gravity pendulum: required number of sensors for different T, k and l. (**A**) Training MSE (square symbols) and test MSE (circle symbols) decrease as the number of sensors increases in the case $k = 1$, $T = 1$ and $l = 0.2$. Training and test MSE versus the number of sensors in different conditions of (**B**) T, (**C**) l, and (**D**) k. For clarity, the standard deviation is not shown. The arrow indicates where the rate of the error decay diminishes.

D.8.2 Error tendency and convergence

Here, we investigate the error tendency under different conditions, including network size, and training dataset size. We take $T = 3$ for the following experiments in this subsection. By varying the network width, we can observe that there is a best width to achieve the smallest error (Fig. D.6A). It is reasonable that increasing the width from 1 to 100 would decrease the error, but the error would instead increase when the width further increases. This could be due to the increased optimization error, and a better learning rate may be used to find a better network. To examine the effect of training dataset size, we choose networks of width 100 to eliminate the network size effect, and the training and test errors using different dataset size are shown in Fig. D.6B. The test and generalization errors of networks with width 50 and 200 are shown in Fig. 6.3.

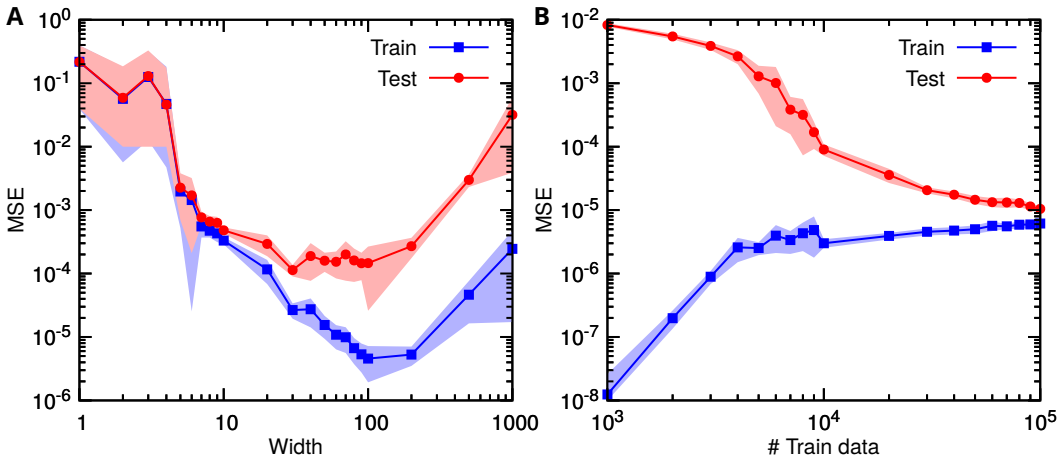


Figure D.6: Gravity pendulum: error tendency and convergence. (**A**) Training and test errors first decrease and then increase, when network width increases. (**B**) Test error decreases when more training data are used (width 100).

D.8.3 Input function spaces

Next, we investigate the effect of different function spaces, including GRF with different length scale l and the space of Chebyshev polynomials with different number of bases. For a fixed sensor number, we observe that there exists a critical length scale, around where the training and test errors change rapidly, see the arrows in Fig. D.7A. This sharp transition around the critical value is also observed in the space of Chebyshev polynomials with different number of bases (Fig. D.7B). The relations between the critical values and the sensor numbers are

$$l \propto m^{-1} \quad \text{and} \quad \#\text{Bases} \propto \sqrt{m}.$$

The inverse relation between l and m is consistent with the theoretical results in Section D.3 and the computational results in Section D.8.1. Therefore, when the function space complexity increases, one may increase m to capture the functions well.

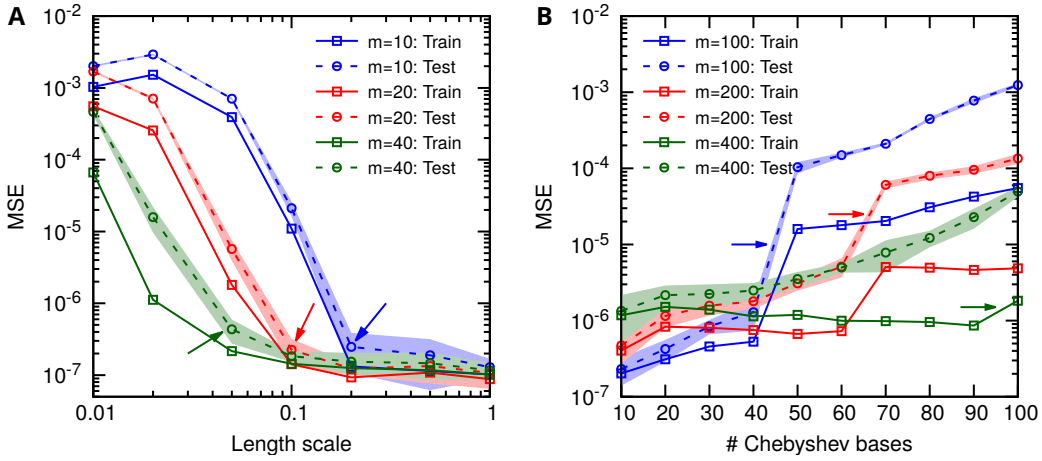


Figure D.7: Gravity pendulum: errors for different input function spaces. Training error (solid line) and test error (dash line) for (**A**) GRF function space with different length scale l. The colors correspond to the number of sensors as shown in the inset. (**B**) Chebyshev polynomials for different number of basis functions.

D.9 Diffusion-reaction system with a source term

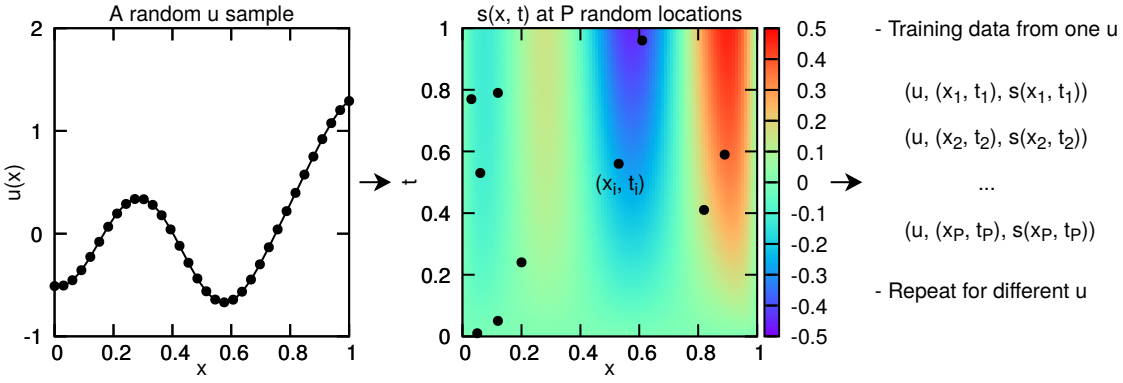


Figure D.8: Learning a diffusion-reaction system. (left) An example of a random sample of the input function $u(x)$. (middle) The corresponding output function $s(x,t)$ at P different (x,t) locations. (right) Pairing of inputs and outputs at the training data points. The total number of training data points is the product of P times the number of samples of u.

The same convergence behavior is also observed when we use $P = \#u$ to generate the training dataset (Fig. D.10), instead of fixing P or $\#u$.

Table D.4: Learning a diffusion-reaction system. Mean and standard deviation of training error and test error of ResNets from 10 independent runs.

# Residual block	Width	Training error	Test error
1	20	$2.4 \times 10^{-3} \pm 4.8 \times 10^{-4}$	$9.9 \times 10^{-2} \pm 4.8 \times 10^{-2}$
1	50	$2.4 \times 10^{-4} \pm 2.6 \times 10^{-5}$	$4.3 \times 10^{-1} \pm 1.3 \times 10^{-1}$
1	100	$2.2 \times 10^{-5} \pm 2.0 \times 10^{-6}$	$1.4 \times 10^{-1} \pm 1.8 \times 10^{-2}$
2	20	$2.2 \times 10^{-3} \pm 2.3 \times 10^{-5}$	$6.0 \times 10^{-1} \pm 1.8 \times 10^{-1}$
2	50	$5.8 \times 10^{-5} \pm 7.3 \times 10^{-6}$	$2.6 \times 10^{-1} \pm 5.6 \times 10^{-2}$
3	20	$1.1 \times 10^{-3} \pm 1.3 \times 10^{-4}$	$6.1 \times 10^{-1} \pm 3.0 \times 10^{-1}$
3	50	$2.8 \times 10^{-5} \pm 3.6 \times 10^{-6}$	$2.1 \times 10^{-1} \pm 6.4 \times 10^{-2}$

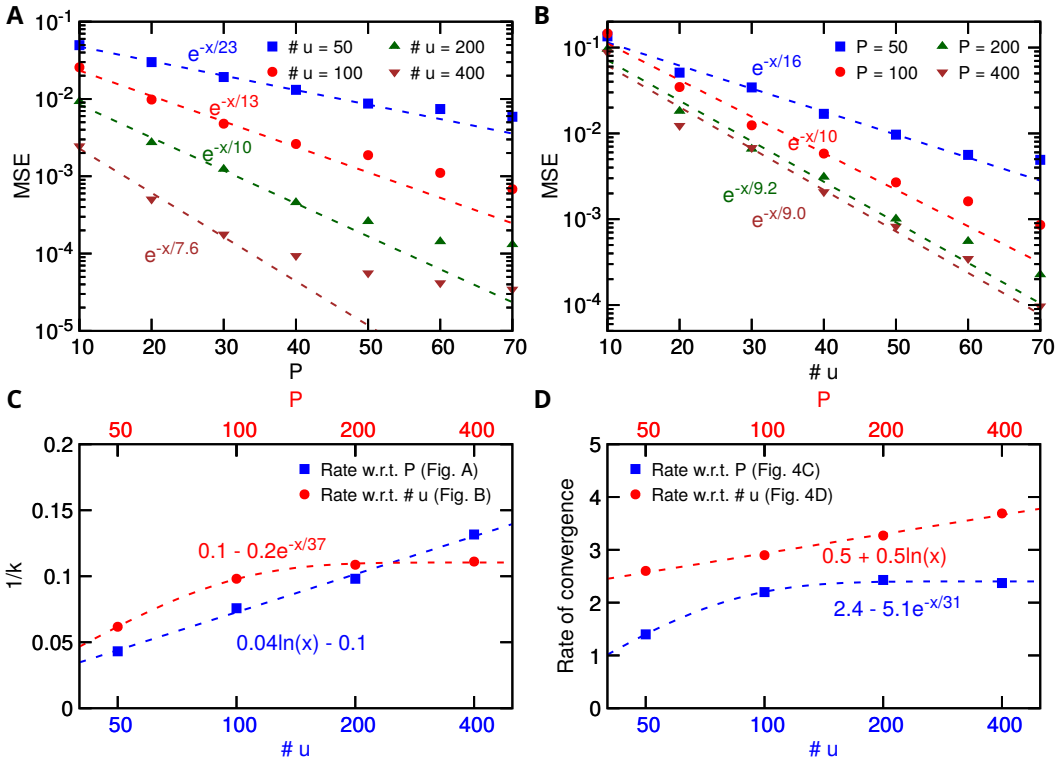


Figure D.9: Diffusion-reaction system: error convergence rates for different number of training data points. (**A**) Exponential convergence of test error with respect to P for different number of u samples. (**B**) Exponential convergence of test error with respect to the number of u samples for different values of P. (**C**) The coefficient $1/k$ in the exponential convergence $e^{-x/k}$ versus the number of u samples or the values of P. (**D**) The polynomial rates of convergence versus the number of u samples or the values of P.

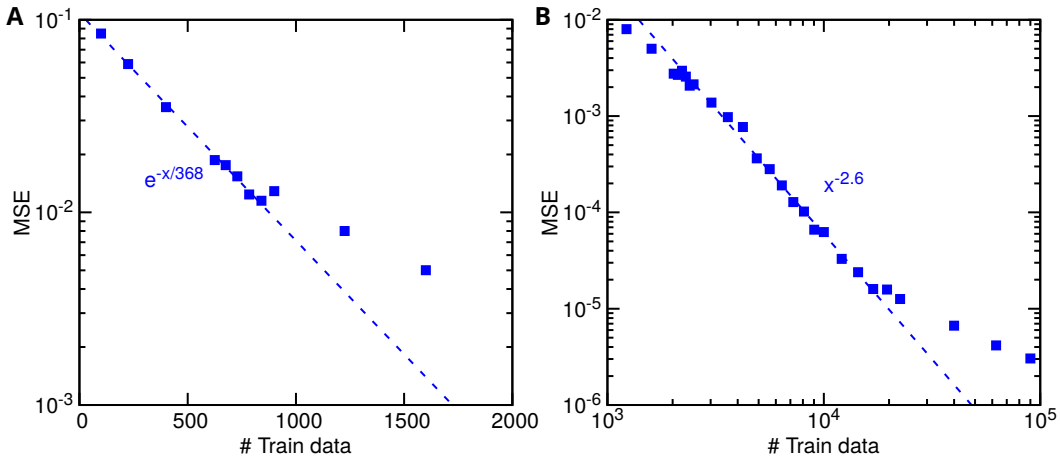


Figure D.10: Diffusion-reaction system: error convergence rates for different number of training data points when $P = \#u$. (**A**) Exponential convergence and (**B**) polynomial convergence.

D.10 Advection equation

Consider the advection equation

$$\textbf{Problem 8} \quad \frac{\partial s}{\partial t} + a(x)\frac{\partial s}{\partial x} = 0, \quad x \in [0,1], t \in [0,1],$$

and here we learn four operators mapping from different input functions to the solution $s(x,t)$. The choices of $a(x)$, IC/BC and DeepONet input function are listed in Table D.5. We note that in these four cases, $a(x)$ is positive. In Cases I and IV we use a periodic BC, and thus the IC is also chosen to be periodic.

To generate the training dataset, we solve the system using a finite difference method on a 100 by 100 grid, except for the Case I with the analytical solution $s(x,t) = f(\sin^2(\pi(x-t))$. Then for each solution $s(x,t)$ we randomly select $P = 100$ points out of these 10000$(= 100 \times 100)$ grid points. The dataset size is equal to the product of P by the number of input function samples. We sample $f(x)$ in Table D.5 from a GRF with the Gaussian kernel of different length scale l, and we use 500 random f samples for training.

Table D.5: Setup and results of four different cases of the advection equation. Mean and standard deviation of the test MSE error are obtained from 10 independent runs.

Case	$a(x)$	IC $s(x,0)$	BC	Input	Length scale l	Test MSE	Example
I	1	$f(\sin^2(\pi x))$	Periodic	$s(x,0)$	0.5	$1.8\times10^{-6}\pm4.2\times10^{-7}$	Fig. D.11
II	1	$xf(x)$	$s(0,t)=0$	$s(x,0)$	0.2	$3.7\times10^{-7}\pm1.8\times10^{-7}$	Fig. D.12
III	$1+0.1\cdot f(x)$	x^2	$s(0,t)=\sin(\pi t)$	$f(x)$	0.2	$1.6\times10^{-5}\pm6.8\times10^{-6}$	Fig. D.13
IV	$1+0.1\cdot\frac{f(x)+f(1-x)}{2}$	$\sin(2\pi x)$	Periodic	$f(x)$	0.2	$3.2\times10^{-5}\pm1.4\times10^{-5}$	Fig. D.14

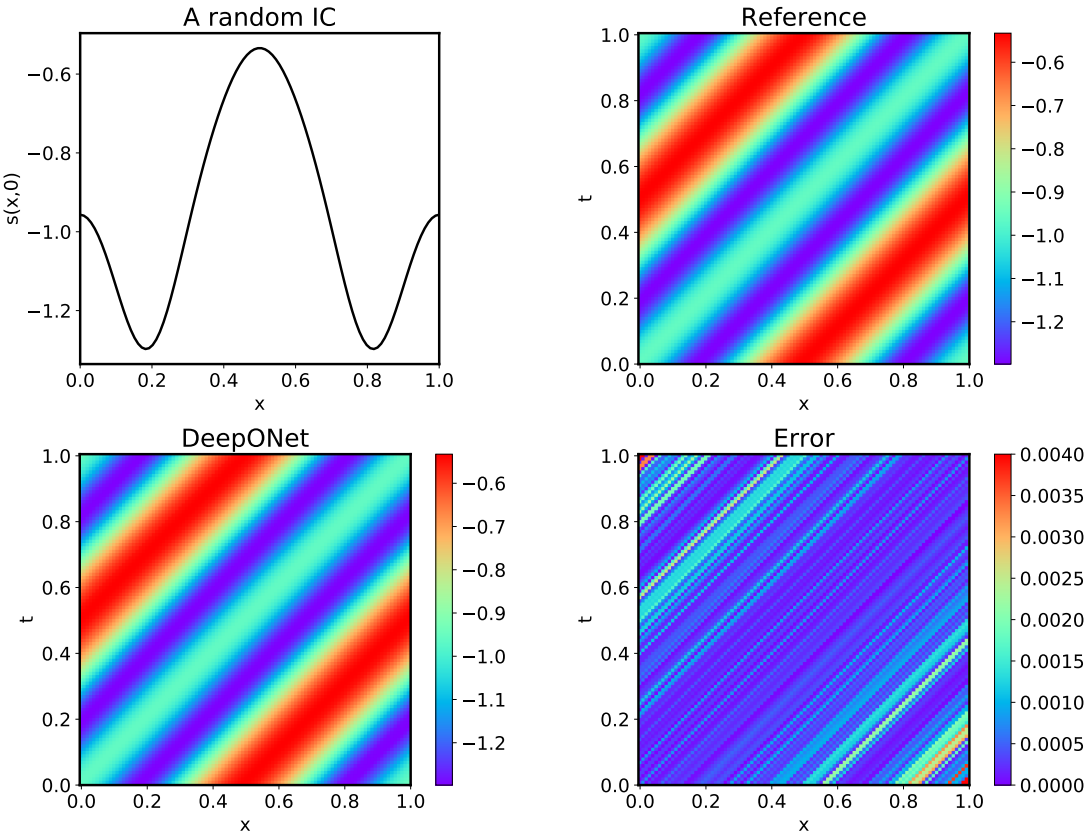


Figure D.11: DeepONet prediction of the advection equation in Case I for a random IC.

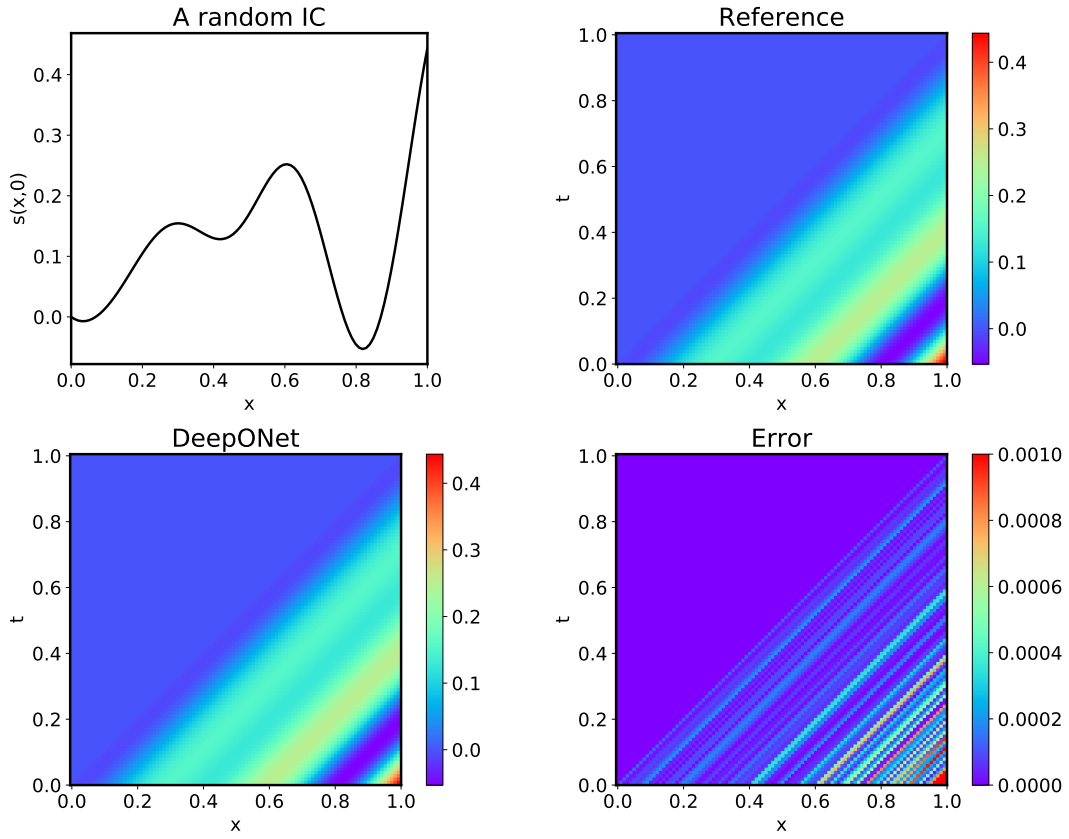


Figure D.12: DeepONet prediction of the advection equation in Case II for a random IC.

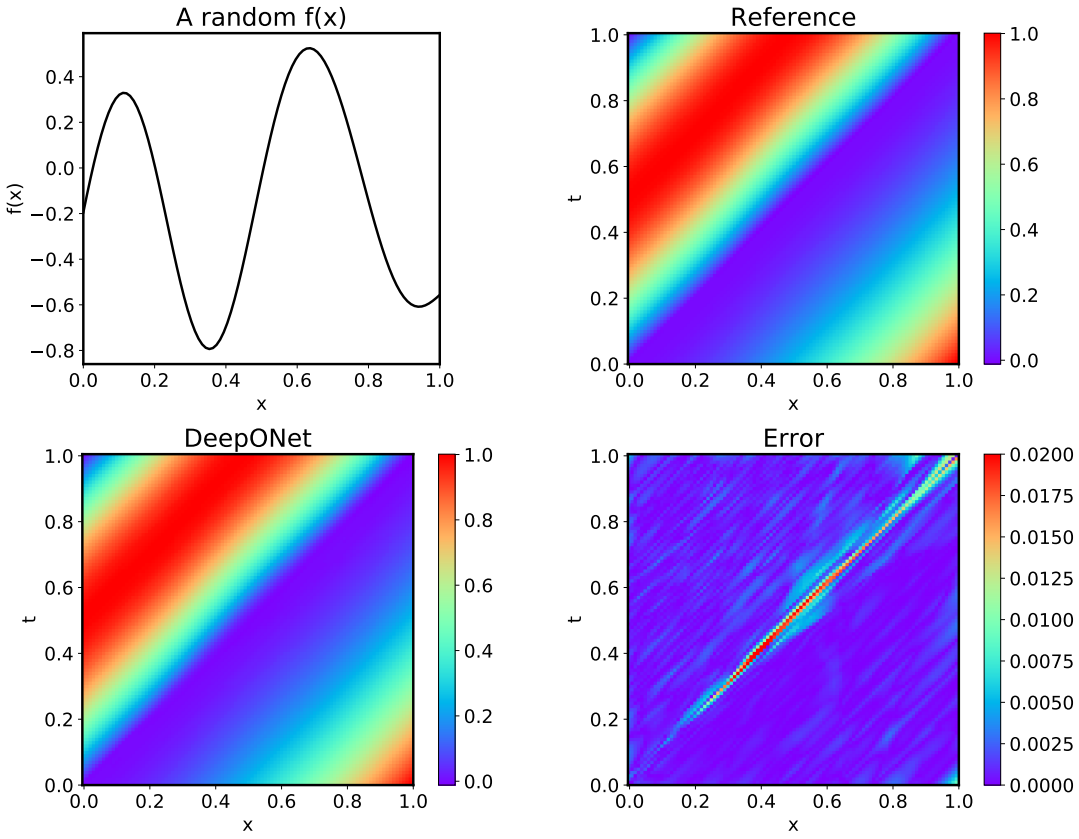


Figure D.13: DeepONet prediction of the advection equation in Case III for a random $f(x)$.

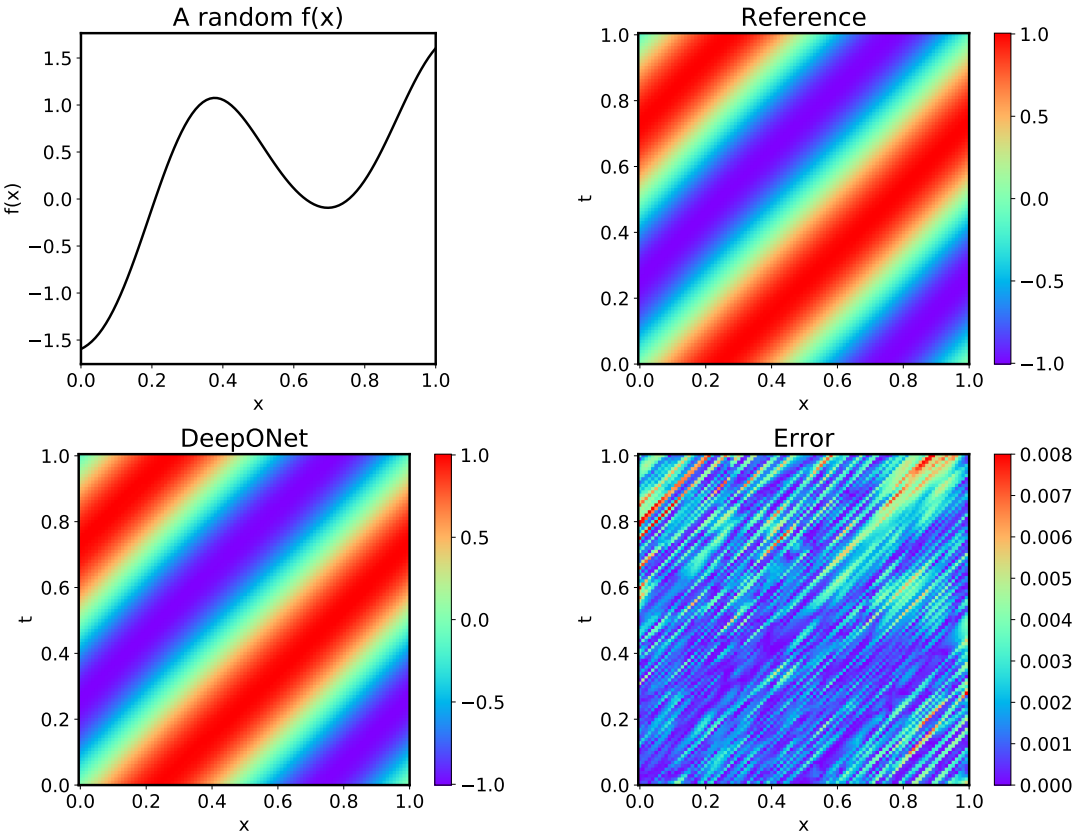


Figure D.14: DeepONet prediction of the advection equation in Case IV for a random $f(x)$.

D.11 Advection-diffusion equation

Consider the advection-diffusion equation

Problem 9
$$\frac{\partial s}{\partial t} + \frac{\partial s}{\partial x} - D\frac{\partial^2 s}{\partial x^2} = 0, \quad x \in (0,1), t \in (0,1], \tag{D.2}$$

with the periodic boundary condition and initial condition $s(x,0) = f(\sin^2(\pi x))$, where $D = 0.1$ is the diffusion coefficient and $f(x)$ is sampled from a GRF with the Gaussian kernel of the length scale $l = 0.5$. Here, $s(x,0)$ is the input function, and we use DeepONets to learn the operator mapping from $s(x,0)$ to the solution $s(x,t)$. To generate the training dataset, we solve the system using a finite difference method on a 100 by 100 grid, and then for each solution $s(x,t)$ we randomly select $P = 100$ points out of these 10000$(= 100 \times 100)$ grid points. The dataset size is equal to the product of P by the number of $s(x,0)$ samples which is set to 500 here. The test MSE is $8.7 \times 10^{-6} \pm 1.1 \times 10^{-6}$ and Fig. D.15 is an example for prediction.

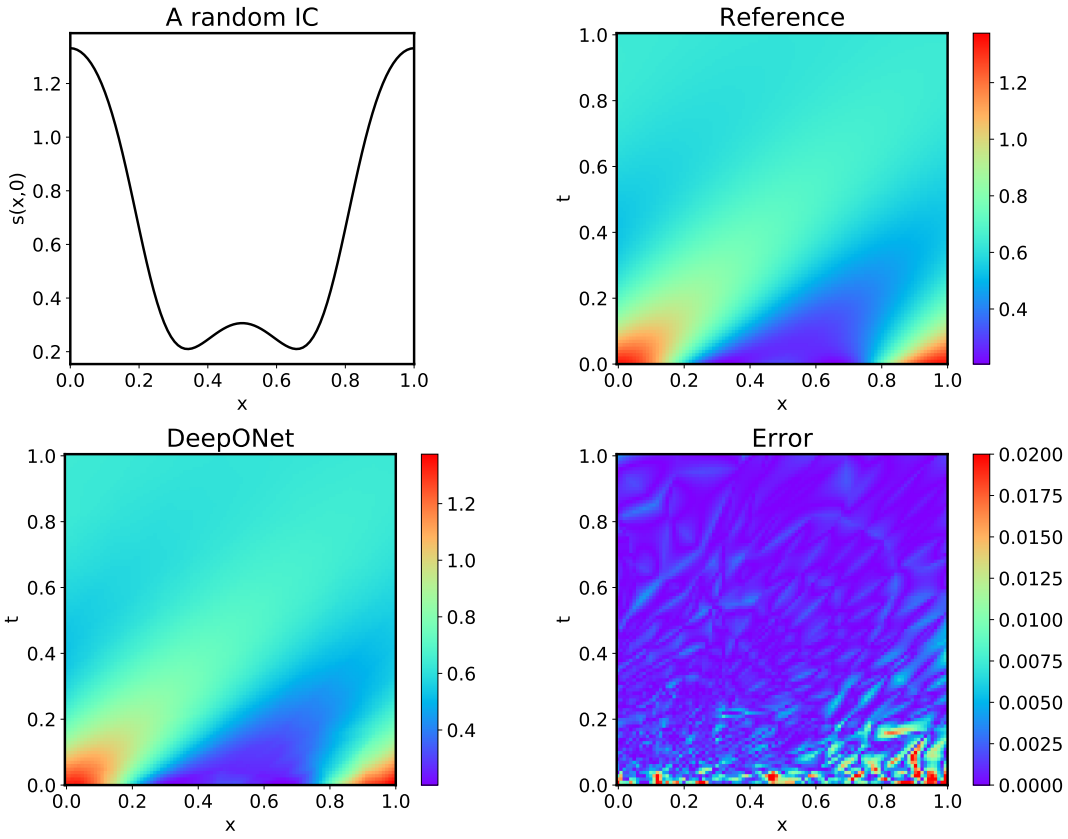


Figure D.15: DeepONet prediction of the advection-diffusion equation for a random IC.

D.12 Fractional differential operators

Below we detail the data generation and NN architectures for learning the 1D Caputo fractional derivative and the 2D fractional Laplacian.

D.12.1 1D Caputo derivative operator

The V spaces we consider are Legendre polynomial space, poly-fractonomial space, and GRF, all of which have been mentioned in Section D.2. For the Legendre and poly-fractonomial spaces, we take $M = 2$ and $N = 11$. The expansion coefficients are taken from Sobol sequence, a quasi-random sequence. For the GRF, we set $l = 0.1$ and $l = 0.3$ in the Gaussian kernel for training and test sets, respectively.

Given the discrete version of the input function $u(\cdot)$, i.e., $[u(x_1), \cdots, u(x_m)]^T$, the evaluation point y, and the fractional order α, the reference solution $G(u)(y, \alpha)$ is computed by using the $L1$ scheme [205], a finite difference scheme for approximating Caputo fractional derivative. We take 2000 finite difference points in order to ensure that the computed value of output function is sufficiently accurate.

The training set is determined by the following parameters. The number of different samples of $u(\cdot)$ is $N_u = 10000$. The number of the equi-spaced "sensors" for $u(\cdot)$ is $m = 15$. The numbers of different equi-spaced evaluation points y and α for the output function $G(u)(y, \alpha)$ are both $N_y = N_\alpha = 10$. The number of training points is calculated as $N_u \times N_y \times N_\alpha = 10^6$. The same parameters are taken for the test set.

D.12.2 2D fractional Laplacian operator

The V space, where $u(r, \theta)$ is sampled from, is an orthogonal polynomial space spanned by the Zernike polynomial $Z_i(r, \theta)$, where r and θ are the radius and angle in the polar coordinate, respectively. We take 15 basis functions $\{Z_0, Z_1, \cdots, Z_{14}\}$, namely, $N = 15$, and set $M = 2$ for sampling the expansion coefficients a_i in Section D.2.

Given the discrete version of the input function $u(\cdot)$, i.e., $[u(x_1), \cdots, u(x_m)]^T$, the evaluation point y, and the fractional order α, the reference solution $G(u)(y, \alpha)$ is computed by using the vector Grünwald-Letnikov formula [145] for approximating Riesz fractional Laplacian. We take 16 Gauss quadrature points for evaluating the integral with respect to differentiation direction and take 2000 finite difference points to approximate fractional directional derivative along a specific differentiation direction.

For DeepONet and CNN models, the raw data are the same. We choose $N_u = 5000$, $N_x = 15 \times 15 = 225$, $N_y = 15 \times 15 = 225$, and $N_\alpha = 10$. But the raw data are organized in different ways for the two models. The DeepONet model organizes the data in a loose way, and thus the size of training set is large, namely, $N_u \times N_y \times N_\alpha = 1.125 \times 10^7$. In contrast, the CNN model leverages the spatial structure of the raw data and rearranges the data into a compact form (image). The size of training set is only $N_u = 5000$. Also, the parameters for the test set are the same as those for the training one.

For the CNN, the filter size is fixed to be 3×3, and the maxpooling window size is 2×2. The input images are shaped to be a 4D array having the shape $[N_u, H_x, W_x, 1]$, where the height (H_x) and width (W_x) of the image for contour plot of one sample

of $u(\cdot)$ are taken to be $H_x = W_x = \sqrt{N_x}$. The output images are arranged to another 4D array having the shape $[N_u, H_y, W_y, N_\alpha]$, where the height ($H_y$) and width ($W_y$) of the images for contour plots of $\{G(u)(y, \alpha_1), G(u)(y, \alpha_2), \cdots, G(u)(y, \alpha_{N_\alpha}\}$ are taken to be $H_y = W_y = \sqrt{N_y}$. We denote by N_{chan} the number of channels in a convolution layer. From the input images to the latent vector, we successively have a convolutional layer with $N_{chan} = 32$, a maxpooling layer, a convolutional layer with $N_{chan} = 64$, and a dense layer with width 64. We take zero padding and stride size 1 for all the above layers. From the latent vector to the output images, we successively have a dense layer with width 64, a dense layer with width 1024 (then reshaped to the shape $[N_u, 4, 4, 64]$), a convolutional layer with $N_{chan} = 32$, padding size 2, and stride 1 (then upsampled to the shape $[N_u, 13, 13, 1]$), and a convolutional layer with $N_{chan} = N_\alpha$, padding size 2 and stride size 1. The CNN architecture we just mentioned are rather similar to encoder and decoder in an autoencoder. We borrow the idea of latent space in the autoencoder, and link the two parts in the CNN using the latent space. The dimensionality of the latent space is set to be 20 in the current example.

D.13 Hölder continuity of the operators in this work

For all the explicit and implicit operators in our examples, the operators G are Hölder continuous.

$$\left\|G(f) - G(g)\right\|_X \leq C \left\|f - g\right\|_Y^\alpha, \quad 0 < \alpha \leq 1.$$

Here $C > 0$ depends on f and g and the operator G. Here we X and Y are Banach spaces and they refers to the space of continuous functions on a compact set unless otherwise stated.

Let $G_{\mathbb{N}}$ be the approximated G using DeepONet. Let f_h be an approximation of f in Y, possibly by collocation or neural networks. Then

$$\left\|G(f) - G_{\mathbb{N}}(f_h)\right\|_X \quad \leq \quad \left\|G(f) - G(f_h)\right\|_X + \left\|G(f_h) - G_{\mathbb{N}}(f_h)\right\|_X \leq C \left\|f - f_h\right\|_Y^\alpha + \varepsilon,$$

where ε is the user-defined accuracy as in the universal approximation theorem by neural networks and thus the key is to verify the operator G is Hölder continuous.

The explicit operators and their Lipschitz continuity ($\alpha = 1$) are presented below.

1. (Simple ODE, **Problem 1.A**) $G(u)(x) = s_0 + \int_0^x u(s)\, ds$.

$$\max_{x\in[0,b]} |G(u)(x) - G(v)(x)| \leq b \max_{x\in[0,b]} |u - v|.$$

2. (Caputo derivative, **Problem 2**) The operator is Lipschitz continuous with respect to its argument in weighted Sobolev norms, c.f. [239, Theorem 2.4].

3. (Integral fractional Laplacian, **Problem 3**) The operator is Lipschitz continuous with respect to its argument in weighted Sobolev norms, c.f. [83, Theorem 3.3].

4. (Legendre transform, **Problem 7** in Equation (D.1)). The Lipschitz continuity of the operator $G(u)(n) = \int_{-1}^{1} P_n(x)u(x)\,dx$ can be seen as follows. For any non-egative integer,

$$\max_n |G(u)(n) - G(v)(n)| \le \max_n \int_{-1}^{1} |P_n(x)|\ dx \max_{x\in[-1,1]} |u - v|$$
$$\le C \max_{x\in[-1,1]} |u - v|$$

   where $C = \max_n \left(\int_{-1}^{1} dx\right)^{1/2}\left(\int_{-1}^{1} |P_n(x)|^2\ dx\right)^{1/2} = \max_n (2\frac{2}{2n+1})^{1/2} \le 2$.

5. The linear operator from **Problem 9** in (D.2) in Lipschitz continuous with respect to the initial condition in the norm in the space of continuous functions, from he classical theory for linear parabolic equations [118, Chapter IV],

The implicit operators from dynamic systems and differential equations are more complicated. The Hölder continuity of these operators are elaborated in the following text.

The implicit operator from **Problem 1**. The Lipschitz continuity of the operator has been shown in the proof of Theorem 6.2.2. For Problem 1.C, the solution is still bounded as long as the input u is not too large. With a bounded solution s, the nonlinear term s^2 is Lipschitz continuous, and thus, the Lipschitz continuity of the operator is still valid, as in the proof of Theorem 6.2.2.

The implicit operator from **Problem 4** is Lipschitz with respect to the right-hand side u of not large values, from the classical theory for quasi-linear parabolic equations [118, Chapter V].

The implicit operator from **Problem 5**. $dy(t,\omega) = k(t,\omega)y(t,\omega)\,dt, \quad y(0) = y_0$. Here $k(t)$ is a Gaussian process with mean zero and a smooth covariance function. The exact solution is $y = y_0 \exp(\int_0^t k(s)\,ds)$. Then the solution operator is Lipschitz continuous in the pathwise sense. Let $k_1(t)$ and $k_2(t)$ be the coefficients in Problem 5 and $y_1(t)$ and $y_2(t)$ are the corresponding solutions. By the mean value theorem, $|e^x - e^y| \le \left|x-y\right|(e^x + e^y)$ holds for all $x, y \in \mathbb{R}$ and

$$\begin{aligned}
\left|y_1(t) - y_2(t)\right| &= \left|y_0\right| \left|\exp(\int_0^t k_1(s)\,ds) - \exp(\int_0^t k_2(s)\,ds)\right| \\
&\le \left|y_0\right| (\exp(\int_0^t k_1(s)\,ds) + \exp(\int_0^t k_2(s)\,ds)) \left|\int_0^t k_1(s) - k_2(s)\,ds\right| \\
&\le \left|y_0\right| \exp(\int_0^t k_1(s)\,ds) + \exp(\int_0^t k_2(s)\,ds)\,|k_1 - k_2|_{C([0,T])}
\end{aligned}$$

Then for any $t \in [0,T]$,

$$\|G(k_1)(\cdot,\omega) - G(k_2)(\cdot,\omega)\|_{C[0,T]} \le C(\omega,t)\,|k_1(\cdot,\omega) - k_2(\cdot,\omega)|_{L^\infty([0,t])},$$

where $C(\omega,T) = \left|y_0\right| (\max_{t\in[0,T]} \exp(\int_0^t k_1(s,\omega)\,ds) + \max_{t\in[0,T]} \exp(\int_0^t k_2(s,\omega)\,ds))$ has moments of any order, according to [33, Proposition 2.3].

The implicit operator from **Problem 6**. The operator can be written as $u = G(b)$ and $u_i = G(b_i)$, $i = 1,2$ satisfy the following equations:

$$-\mathrm{div}(e^{b_i(x)}\nabla u_i) = f(x), \quad x \in D = (0,1), \qquad u_i(x) = 0, \quad x \in \partial D,$$

Then $\|u_i(\omega)\|_{H^1} \leq (\min_{x\in D} e^{-b_i})^{-1} \left\|f\right\|_{H^{-1}}$, where H^1 and H^{-1} are standard Sobolev-Hilbert spaces. The difference $u_1 - u_2$ satisfies the following

$$-\mathrm{div}(e^{b_1(x)}\nabla(u_1-u_2)) = \mathrm{div}((e^{b_1(x)}-e^{b_2(x)})\nabla u_2), \quad x \in D, \qquad u_1-u_2 = 0, \quad x \in \partial D.$$

Then by the stability of the elliptic equation , we have

$$\begin{aligned}
\|u_1(\omega) - u_2(\omega)\|_{H^1} &\leq (\min_{x\in D} e^{b_i(x)})^{-1} \left\|(e^{b_1} - e^{b_2})\nabla u_2\right\|_{L^2} \\
&= (\min_{x\in D} e^{b_1(x)})^{-1} \left\|e^{b_1} - e^{b_2}\right\|_{C(D)} \|u_2\|_{H^1} \\
&\leq (\min_{x\in D} e^{b_1(x)})^{-1}(\min_{x\in D} e^{b_2(x)})^{-1} \left\|e^{b_1} - e^{b_2}\right\|_{C(D)} \left\|f\right\|_{H^{-1}}
\end{aligned}$$

Then by the mean value theorem, $|e^x - e^y| \leq \left|x - y\right| (e^x + e^y)$ holds for all $x, y \in \mathbb{R}$. Thus,

$$\begin{aligned}
&\|u_1(\omega) - u_2(\omega)\|_{H_0^1} \\
&\quad \leq (\min_{x\in D} e^{b_1(x)})^{-1}(\min_{x\in D} e^{b_2(x)})^{-1}(\left\|e^{b_1}\right\|_{C(D)} + \left\|e^{b_2}\right\|_{C(D)}) \|b_1 - b_2\|_{C(D)} \left\|f\right\|_{H^{-1}}.
\end{aligned}$$

According to [33, Proposition 2.3], all the random variables $\min_{x\in D} e^{b_i(x)})^{-1}$ and $\left\|e^{b_i}\right\|_{C(D)}$, $i = 1, 2$ have any moments of finite order. Then, we obtain the pathwise Lipschitz continuity of the operator G

$$\|\mathcal{G}(b_1)(\omega) - \mathcal{G}(b_2)(\omega)\|_{H^1} \leq C(\omega) \|b_1 - b_2\|_{C(D)}.$$

Here $C(\omega) = (\min_{x\in D} e^{b_1(x)})^{-1}(\min_{x\in D} e^{b_2(x)})^{-1}(\left\|e^{b_1}\right\|_{C(D)} + \left\|e^{b_2}\right\|_{C(D)}) \left\|f\right\|_{H^{-1}}$.

The implicit operator in **Problem 8** is also Lipschitz continuous. The solution

can be represented as $s_0(X^{-1}(t,x))$, where $X(t,x)$ satisfies the following equation

$$\dot{X}(t,x) = a(t, X(t,x)), \quad X(0,x) = x.$$

As there exists $a_0 > 0$ such that $a(t,x) > a_0 > 0$ in the setting of Problem 8, $X(t,x) > 0$ is away from zero. Denote the solution with coefficient a and initial condition u_0 by $u_0(X_a^{-1})$ and the solution with coefficient b and initial condition v_0 by $v_0(X_b^{-1})$. We then can derive the Lipschitz continuity of the operator from the following facts that

$$\begin{aligned} &u_0(X_a^{-1}(t,x)) - v_0(X_b^{-1})(t,x) \\ &\qquad = \left(u_0(X_a^{-1}(t,x)) - u_0(X_b^{-1}(t,x))\right) + \left(u_0(X_b^{-1}(t,x)) - v_0(X_b^{-1})(t,x)\right) \end{aligned}$$

and that u_0, v_0 are Lipschitz and X_a and X_a^{-1} (X_b and X_b^{-1}) are Lipchitz continuous with respect to a (b).

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. TensorFlow: A system for large-scale machine learning. In *12th USENIX Symposium on Operating Systems Design and Implementation*, pages 265–283, 2016.

[2] A. F. Agarap. Deep learning using rectified linear units (ReLU). *arXiv preprint arXiv:1803.08375*, 2018.

[3] M. Ainsworth and J. T. Oden. *A posteriori error estimation in finite element analysis*, volume 37. John Wiley & Sons, 2011.

[4] Z. Allen-Zhu, Y. Li, and Y. Liang. Learning and generalization in over-parameterized neural networks, going beyond two layers. *arXiv preprint arXiv:1811.04918*, 2018.

[5] Z. Allen-Zhu, Y. Li, and Z. Song. A convergence theory for deep learning via over-parameterization. *arXiv preprint arXiv:1811.03962*, 2018.

[6] S. Amari, H. Park, and T. Ozeki. Singularities affect dynamics of learning in neuromanifolds. *Neural Computation*, 18(5):1007–1065, 2006.

[7] S. Arora, N. Cohen, N. Golowich, and W. Hu. A convergence analysis of gradient descent for deep linear neural networks. *arXiv preprint arXiv:1810.02281*, 2018.

[8] S. Arora, S. Du, W. Hu, Z. Li, and R. Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. *arXiv preprint arXiv:1901.08584*, 2019.

[9] S. Arora, R. Ge, B. Neyshabur, and Y. Zhang. Stronger generalization bounds for deep nets via a compression approach. *arXiv preprint arXiv:1802.05296*, 2018.

[10] J. L. Ba, J. R. Kiros, and G. E. Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

[11] A. Baeza and P. Mulet. Adaptive mesh refinement techniques for high-order shock capturing schemes for multi-dimensional hydrodynamic simulations. *International Journal for Numerical Methods in Fluids*, 52(4):455–471, 2006.

[12] N. Baker, F. Alexander, T. Bremer, A. Hagberg, Y. Kevrekidis, H. Najm, M. Parashar, A. Patra, J. Sethian, S. Wild, et al. Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence. Technical report, US DOE Office of Science, Washington, DC (United States), 2019.

[13] P. Bartlett, D. Foster, and M. Telgarsky. Spectrally-normalized margin bounds for neural networks. In *Advances in Neural Information Processing Systems*, pages 6240–6249, 2017.

[14] P. Bartlett, N. Harvey, C. Liaw, and A. Mehrabian. Nearly-tight VC-dimension and pseudodimension bounds for piecewise linear neural networks. *arXiv preprint arXiv:1703.02930*, 2017.

[15] P. Bartlett and S. Mendelson. Rademacher and Gaussian complexities: Risk bounds and structural results. *Journal of Machine Learning Research*, 3(Nov):463–482, 2002.

[16] A. G. Baydin, B. A. Pearlmutter, A. A. Radul, and J. M. Siskind. Automatic differentiation in machine learning: a survey. *Journal of Machine Learning Research*, 18(1):5595–5637, 2017.

[17] C. Baykal, L. Liebenwein, I. Gilitschenski, D. Feldman, and D. Rus. Data-dependent coresets for compressing neural networks with applications to generalization bounds. *arXiv preprint arXiv:1804.05345*, 2018.

[18] C. Beck, W. E, and A. Jentzen. Machine learning approximation algorithms for high-dimensional fully nonlinear partial differential equations and second-order backward stochastic differential equations. *Journal of Nonlinear Science*, pages 1–57, 2017.

[19] M. Belkin, D. Hsu, S. Ma, and S. Mandal. Reconciling modern machine learning and the bias-variance trade-off. *arXiv preprint arXiv:1812.11118*, 2018.

[20] J. Berg and K. Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

[21] M. Betancourt. A geometric theory of higher-order automatic differentiation. *arXiv preprint arXiv:1812.11592*, 2018.

[22] J. Bettencourt, M. J. Johnson, and D. Duvenaud. Taylor-mode automatic differentiation for higher-order derivatives in JAX. 2019.

[23] A. Blum and R. L. Rivest. Training a 3-node neural network is NP-complete. In *Advances in Neural Information Processing Systems*, pages 494–501, 1989.

[24] M. Born and E. Wolf. *Principles of optics: electromagnetic theory of propagation, interference and diffraction of light*. Elsevier, 2013.

[25] L. Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[26] L. Bottou and O. Bousquet. The tradeoffs of large scale learning. In *Advances in Neural Information Processing Systems*, pages 161–168, 2008.

[27] S. L. Brunton, J. L. Proctor, and J. N. Kutz. Discovering governing equations from data by sparse identification of nonlinear dynamical systems. *Proceedings of the National Academy of Sciences*, 113(15):3932–3937, 2016.

[28] J.-L. Bucaille, S. Stauss, E. Felder, and J. Michler. Determination of plastic properties of metals by instrumented indentation using different sharp indenters. *Acta Materialia*, 51(6):1663–1678, 2003.

[29] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[30] Y. Cao and Q. Gu. A generalization theory of gradient descent for learning over-parameterized deep ReLU networks. *arXiv preprint arXiv:1902.01384*, 2019.

[31] Y. P. Cao and J. Lu. Depth-sensing instrumented indentation with dual sharp indenters: stability analysis and corresponding regularization schemes. *Acta Materialia*, 52(5):1143–1153, 2004.

[32] Y. P. Cao and J. Lu. A new method to extract the plastic properties of metal materials from an instrumented spherical indentation loading curve. *Acta Materialia*, 52(13):4023–4032, 2004.

[33] J. Charrier. Strong and weak error estimates for elliptic partial differential equations with random coefficients. *SIAM Journal on Numerical Analysis*, 50(1):216–246, 2012.

[34] M. Chen, J. Pennington, and S. Schoenholz. Dynamical isometry and a mean field theory of RNNs: Gating enables signal propagation in recurrent neural networks. In *International Conference on Machine Learning*, pages 872–881, 2018.

[35] T. Chen and H. Chen. Approximations of continuous functionals by neural networks with application to dynamic systems. *IEEE Transactions on Neural Networks*, 4(6):910–918, 1993.

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

[37] T. Chen and H. Chen. Universal approximation to nonlinear operators by neural networks with arbitrary activation functions and its application to dynamical systems. *IEEE Transactions on Neural Networks*, 6(4):911–917, 1995.

[38] T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud. Neural ordinary differential equations. In *Advances in Neural Information Processing Systems*, pages 6571–6583, 2018.

[39] Y. Chen, C. Jin, and B. Yu. Stability and convergence trade-off of iterative optimization algorithms. *arXiv preprint arXiv:1804.01619*, 2018.

[40] Y. Chen, L. Lu, G. E. Karniadakis, and L. D. Negro. Physics-informed neural networks for inverse problems in nano-optics and metamaterials. *arXiv preprint arXiv:1912.01085*, 2019.

[41] Y. Cheng, D. Wang, P. Zhou, and T. Zhang. Model compression and acceleration for deep neural networks: The principles, progress, and challenges. *IEEE Signal Processing Magazine*, 35(1):126–136, 2018.

[42] Y.-T. Cheng and C.-M. Cheng. Relationships between hardness, elastic modulus, and the work of indentation. *Applied Physics Letters*, 73(5):614–616, 1998.

[43] Y.-T. Cheng and C.-M. Cheng. Scaling, dimensional analysis, and indentation measurements. *Materials Science and Engineering: R: Reports*, 44(4-5):91–149, 2004.

[44] I.-S. Choi, M. Dao, and S. Suresh. Mechanics of indentation of plastically graded materials–I: Analysis. *Journal of the Mechanics and Physics of Solids*, 56(1):157–171, 2008.

[45] I.-S. Choi, O. Kraft, and R. Schwaiger. Validity of the reduced modulus concept to describe indentation loading response for elastoplastic materials with sharp indenters. *Journal of Materials Research*, 24(3):998–1006, 2009.

[46] N. Chollacoop, M. Dao, and S. Suresh. Depth-sensing instrumented indentation with dual sharp indenters. *Acta Materialia*, 51(13):3713–3729, 2003.

[47] P. G. Ciarlet. *The finite element method for elliptic problems*, volume 40. Siam, 2002.

[48] D. A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv preprint arXiv:1511.07289*, 2015.

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

[50] M. Dao, N. v. Chollacoop, K. Van Vliet, T. Venkatesh, and S. Suresh. Computational modeling of the forward and reverse problems in instrumented sharp indentation. *Acta Materialia*, 49(19):3899–3918, 2001.

[51] A. De Myttenaere, B. Golden, B. Le Grand, and F. Rossi. Mean absolute percentage error for regression models. *Neurocomputing*, 192:38–48, 2016.

[52] L. Dinh, R. Pascanu, S. Bengio, and Y. Bengio. Sharp minima can generalize for deep nets. In *International Conference on Machine Learning*, pages 1019–1028. JMLR. org, 2017.

[53] M. Dissanayake and N. Phan-Thien. Neural-network-based approximations for solving partial differential equations. *Communications in Numerical Methods in Engineering*, 10(3):195–201, 1994.

[54] S. S. Du, C. Jin, J. D. Lee, M. I. Jordan, A. Singh, and B. Poczos. Gradient descent can take exponential time to escape saddle points. In *Advances in Neural Information Processing Systems*, pages 1067–1077, 2017.

[55] S. S. Du, J. D. Lee, H. Li, L. Wang, and X. Zhai. Gradient descent finds global minima of deep neural networks. *arXiv preprint arXiv:1811.03804*, 2018.

[56] S. S. Du, J. D. Lee, Y. Tian, A. Singh, and B. Poczos. Gradient descent learns one-hidden-layer CNN: Don't be afraid of spurious local minima. In *International Conference on Machine Learning*, pages 1338–1347, 2018.

[57] J. Duchi, E. Hazan, and Y. Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159, 2011.

[58] R. Dudley. Balls in $\mathbb{R}^k$ do not cut all subsets of $k + 2$ points. *Advances in Mathematics*, 31(3):306 – 308, 1979.

[59] V. Dumoulin, E. Perez, N. Schucher, F. Strub, H. d. Vries, A. Courville, and Y. Bengio. Feature-wise transformations. *Distill*, 2018. https://distill.pub/2018/feature-wise-transformations.

[60] G. Dziugaite and D. Roy. Computing nonvacuous generalization bounds for deep (stochastic) neural networks with many more parameters than training data. *arXiv preprint arXiv:1703.11008*, 2017.

[61] W. E and B. Yu. The deep Ritz method: A deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

[62] N. B. Erichson, M. Muehlebach, and M. W. Mahoney. Physics-informed autoencoders for Lyapunov-stable fluid flow prediction. *arXiv preprint arXiv:1905.10866*, 2019.

[63] A. Esteva, A. Robicquet, B. Ramsundar, V. Kuleshov, M. DePristo, K. Chou, C. Cui, G. Corrado, S. Thrun, and J. Dean. A guide to deep learning in healthcare. *Nature Medicine*, 25(1):24–29, 2019.

[64] C. Finn, P. Abbeel, and S. Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, pages 1126–1135, 2017.

[65] A. Forrester, A. Sobester, and A. Keane. *Engineering design via surrogate modelling: a practical guide*. John Wiley & Sons, 2008.

[66] J. Friedman, T. Hastie, and R. Tibshirani. *The elements of statistical learning*. Springer series in statistics New York, 2001.

[67] K. Fukumizu and S. Amari. Local minima and plateaus in hierarchical structures of multilayer perceptrons. *Neural Networks*, 13(3):317–327, 2000.

[68] R. Ge, F. Huang, C. Jin, and Y. Yuan. Escaping from saddle points—online stochastic gradient for tensor decomposition. In *Conference on Learning Theory*, pages 797–842, 2015.

[69] R. Ge, J. D. Lee, and T. Ma. Matrix completion has no spurious local minimum. In *Advances in Neural Information Processing Systems*, pages 2973–2981, 2016.

[70] A. Giannakopoulos and S. Suresh. Determination of elastoplastic properties by instrumented sharp indentation. *Scripta Materialia*, 40(10):1191–1198, 1999.

[71] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[72] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[73] A. Gonen and S. Shalev-Shwartz. Fast rates for empirical risk minimization of strict saddle problems. *arXiv preprint arXiv:1701.04271*, 2017.

[74] A. Gouldstone, N. Chollacoop, M. Dao, J. Li, A. M. Minor, and Y.-L. Shen. Indentation across size scales and disciplines: Recent developments in experimentation and modeling. *Acta Materialia*, 55(12):4015–4039, 2007.

[75] P. Grohs, F. Hornung, A. Jentzen, and P. Von Wurstemberger. A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of black-scholes partial differential equations. *arXiv preprint arXiv:1809.02362*, 2018.

[76] M. Gulian, M. Raissi, P. Perdikaris, and G. Karniadakis. Machine learning of space-fractional differential equations. *SIAM Journal on Scientific Computing*, 41(4):A2485–A2509, 2019.

[77] S. Gunasekar, J. Lee, D. Soudry, and N. Srebro. Implicit bias of gradient descent on linear convolutional networks. In *Advances in Neural Information Processing Systems*, pages 9461–9471, 2018.

[78] R. Haj-Ali, H.-K. Kim, S. W. Koh, A. Saxena, and R. Tummala. Nonlinear constitutive models from nanoindentation tests using artificial neural networks. *International Journal of Plasticity*, 24(3):371–396, 2008.

[79] J. Han, A. Jentzen, and W. E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[80] B. Hanin. Universal function approximation by deep neural nets with bounded width and ReLU activations. *arXiv preprint arXiv:1708.02691*, 2017.

[81] B. Hanin. Which neural net architectures give rise to exploding and vanishing gradients? In *Advances in Neural Information Processing Systems*, pages 580–589, 2018.

[82] B. Hanin and M. Sellke. Approximating continuous functions by ReLU nets of minimal width. *arXiv preprint arXiv:1710.11278*, 2017.

[83] Z. Hao, H. Li, Z. Zhang, and Z. Zhang. Sharp error estimates of a spectral Galerkin method for a diffusion-reaction equation with integral fractional Laplacian on a disk. *Submitted*, 2019. https://www.researchgate.net/publication/335892990.

[84] M. Hardt, B. Recht, and Y. Singer. Train faster, generalize better: Stability of stochastic gradient descent. *arXiv preprint arXiv:1509.01240*, 2015.

[85] J. He, L. Li, J. Xu, and C. Zheng. ReLU deep neural networks and linear finite elements. *arXiv preprint arXiv:1807.03973*, 2018.

[86] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[87] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

[88] Q. He, D. Brajas-Solano, G. Tartakovsky, and A. M. Tartakovsky. Physics-informed neural networks for multiphysics data assimilation with application to subsurface transport. *arXiv preprint arXiv:1912.02968*, 2019.

[89] G. Hinton. Overview of mini-batch gradient descent. http://www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf, 2014.

[90] G. Hinton, L. Deng, D. Yu, G. E. Dahl, A. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for

acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

[91] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989.

[92] N. Huber, A. Konstantinidis, and C. Tsakmakis. Determination of poisson's ratio by spherical indentation using neural networks—part I: theory. *Journal of Applied Mechanics*, 68(2):218–223, 2001.

[93] N. Huber and C. Tsakmakis. Determination of poisson's ratio by spherical indentation using neural networks—part II: Identification method. *Journal of Applied Mechanics*, 68(2):224–229, 2001.

[94] T. J. Hughes, J. A. Cottrell, and Y. Bazilevs. Isogeometric analysis: Cad, finite elements, nurbs, exact geometry and mesh refinement. *Computer Methods in Applied Mechanics and Engineering*, 194(39-41):4135–4195, 2005.

[95] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, 2015.

[96] R. Iten, T. Metger, H. Wilming, L. Del Rio, and R. Renner. Discovering physical concepts with neural networks. *arXiv preprint arXiv:1807.10300*, 2018.

[97] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks. *arXiv preprint arXiv:1909.12228*, 2019.

[98] A. D. Jagtap, K. Kawaguchi, and G. E. Karniadakis. Adaptive activation functions accelerate convergence in deep and physics-informed neural networks. *Journal of Computational Physics*, 404:109136, 2020.

[99] J. Jia and A. R. Benson. Neural jump stochastic differential equations. *arXiv preprint arXiv:1905.10403*, 2019.

[100] C. Jin, R. Ge, P. Netrapalli, S. M. Kakade, and M. I. Jordan. How to escape saddle points efficiently. In *International Conference on Machine Learning*, volume 70, pages 1724–1732, 2017.

[101] P. Jin, L. Lu, Y. Tang, and G. E. Karniadakis. Quantifying the generalization error in deep learning in terms of data distribution and neural network smoothness. *arXiv preprint arXiv:1905.11427*, 2019.

[102] C. Johnson. *Numerical solution of partial differential equations by the finite element method*. Courier Corporation, 2012.

[103] K. Johnson. *Contact mechanics*. Cambridge University Press, Cambridge, UK, 1985.

[104] O. Jørgensen, A. Giannakopoulos, and S. Suresh. Spherical indentation of composite laminates with controlled gradients in elastic anisotropy. *International Journal of Solids and Structures*, 35(36):5097–5114, 1998.

[105] G. E. Karniadakis and S. J. Sherwin. *Spectral/hp element methods for computational fluid dynamics*. Oxford University Press, second edition, 2013.

[106] Y. Kassahun, B. Yu, A. T. Tibebu, D. Stoyanov, S. Giannarou, J. H. Metzen, and E. Vander Poorten. Surgical robotics beyond enhanced dexterity instrumentation: a survey of machine learning techniques and their role in

intelligent and autonomous surgical actions. *International Journal of Computer Assisted Radiology and Surgery*, 11(4):553–568, 2016.

[107] K. Kawaguchi. Deep learning without poor local minima. In *Advances in Neural Information Processing Systems*, pages 586–594, 2016.

[108] K. Kawaguchi, L. Kaelbling, and Y. Bengio. Generalization in deep learning. *arXiv preprint arXiv:1710.05468*, 2017.

[109] M. C. Kennedy and A. O'Hagan. Predicting the output from a complex computer code when fast approximations are available. *Biometrika*, 87(1):1–13, 2000.

[110] N. Keskar, D. Mudigere, J. Nocedal, M. Smelyanskiy, and P. Tang. On large-batch training for deep learning: Generalization gap and sharp minima. *arXiv preprint arXiv:1609.04836*, 2016.

[111] Y. Khoo, J. Lu, and L. Ying. Solving parametric PDE problems with artificial neural networks. *arXiv preprint arXiv:1707.03351*, 2017.

[112] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. In *International Conference on Learning Representations*, 2015.

[113] G. Klambauer, T. Unterthiner, A. Mayr, and S. Hochreiter. Self-normalizing neural networks. In *Advances in Neural Information Processing Systems*, pages 972–981, 2017.

[114] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[115] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1097–1105, 2012.

[116] P. Kumar, O. Prakash, and U. Ramamurty. Micro-and meso-structures and their influence on mechanical properties of selectively laser melted Ti-6Al-4V. *Acta Materialia*, 154:246–260, 2018.

[117] I. Kuzborskij and C. Lampert. Data-dependent stability of stochastic gradient descent. *arXiv preprint arXiv:1703.01678*, 2017.

[118] O. A. Ladyženskaja, V. A. Solonnikov, and N. N. Ural'ceva. *Linear and quasi-linear equations of parabolic type*. AMS, Providence, R.I., 1968.

[119] I. E. Lagaris, A. Likas, and D. I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[120] I. E. Lagaris, A. C. Likas, and D. G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000.

[121] H. Lan and T. Venkatesh. Determination of the elastic and plastic properties of materials through instrumented indentation with reduced sensitivity. *Acta Materialia*, 55(6):2025–2041, 2007.

[122] M.-Q. Le. Material characterization by instrumented spherical indentation. *Mechanics of Materials*, 46:42–56, 2012.

[123] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 521(7553):436, 2015.

[124] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[125] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural networks: Tricks of the trade*, pages 9–50. Springer, 1998.

[126] J. D. Lee, M. Simchowitz, M. I. Jordan, and B. Recht. Gradient descent only converges to minimizers. In *Conference on Learning Theory*, pages 1246–1257, 2016.

[127] H. Li, L. Gutierrez, H. Toda, O. Kuwazuru, W. Liu, Y. Hangai, M. Kobayashi, and R. Batres. Identification of material properties using nanoindentation and surrogate modeling. *International Journal of Solids and Structures*, 81:151–159, 2016.

[128] Y. Li and Y. Liang. Learning overparameterized neural networks via stochastic gradient descent on structured data. In *Advances in Neural Information Processing Systems*, pages 8157–8166, 2018.

[129] Y. Li, P. Stevens, M. Sun, C. Zhang, and W. Wang. Improvement of predicting mechanical properties from spherical indentation test. *International Journal of Mechanical Sciences*, 117:182–196, 2016.

[130] T. Liang, T. Poggio, A. Rakhlin, and J. Stokes. Fisher-Rao metric, geometry, and complexity of neural networks. *arXiv preprint arXiv:1711.01530*, 2017.

[131] Q. Liao and T. Poggio. Theory II: Landscape of the empirical risk in deep learning. *arXiv preprint arXiv:1703.09833*, 2017.

[132] A. Lischke, G. Pang, M. Gulian, F. Song, C. Glusa, X. Zheng, Z. Mao, W. Cai, M. M. Meerschaert, M. Ainsworth, et al. What is the fractional Laplacian? a comparative review with new results. *Journal of Computational Physics*, 404:109009, 2020.

[133] Z. Long, Y. Lu, X. Ma, and B. Dong. PDE-net: Learning PDEs from data. In *International Conference on Machine Learning*, pages 3214–3222, 2018.

[134] L. Lu, M. Dao, P. Kumar, U. Ramamurty, G. E. Karniadakis, and S. Suresh. Extraction of mechanical properties of materials through deep learning from instrumented indentation. *Proceedings of the National Academy of Sciences*, 117(13):7052–7062, 2020.

[135] L. Lu, P. Jin, and G. E. Karniadakis. DeepONet: Learning nonlinear operators for identifying differential equations based on the universal approximation theorem of operators. *arXiv preprint arXiv:1910.03193*, 2019.

[136] L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis. DeepXDE: A deep learning library for solving differential equations. *arXiv preprint arXiv:1907.04502*, 2019.

[137] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis. Dying ReLU and initialization: Theory and numerical examples. *arXiv preprint arXiv:1903.06733*, 2019.

[138] L. Lu, Y. Su, and G. E. Karniadakis. Collapse of deep and narrow neural nets. *arXiv preprint arXiv:1808.04947*, 2018.

[139] D. Ma, C. W. Ong, J. Lu, and J. He. Methodology for the evaluation of yield strength and hardening behavior of metallic materials by indentation with spherical tip. *Journal of Applied Physics*, 94(1):288–294, 2003.

[140] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *International Conference on Machine Learning*, volume 30, page 3, 2013.

[141] A. Mahmoudi and S. Nourbakhsh. A neural networks approach to characterize material properties using the spherical indentation test. *Procedia Engineering*, 10:3062–3067, 2011.

[142] Z. Mao, A. D. Jagtap, and G. E. Karniadakis. Physics-informed neural networks for high-speed flows. *Computer Methods in Applied Mechanics and Engineering*, 360:112789, 2020.

[143] C. C. Margossian. A review of automatic differentiation and its efficient implementation. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 9(4):e1305, 2019.

[144] A. J. Meade Jr and A. A. Fernandez. The numerical solution of linear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 19(12):1–25, 1994.

[145] M. M. Meerschaert, J. Mortensen, and H.-P. Scheffler. Vector grunwald formula for fractional derivatives. *Fractional Calculus and Applied Analysis*, 7(1):61–82, 2004.

[146] X. Meng and G. E. Karniadakis. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems. *Journal of Computational Physics*, 401:109020, 2020.

[147] X. Meng, Z. Li, D. Zhang, and G. E. Karniadakis. PPINN: Parareal physics-informed neural network for time-dependent pdes. *arXiv preprint arXiv:1909.10145*, 2019.

[148] H. N. Mhaskar and N. Hahm. Neural networks for functional approximation and system identification. *Neural Computation*, 9(1):143–159, 1997.

[149] D. Mishkin and J. Matas. All you need is a good init. In *International Conference on Learning Representations*, 2016.

[150] M. Mitzenmacher and E. Upfal. *Probability and computing: randomization and probabilistic techniques in algorithms and data analysis*. Cambridge university press, 2017.

[151] M. A. Nabian and H. Meidani. A deep neural network surrogate for high-dimensional random partial differential equations. *arXiv preprint arXiv:1806.02957*, 2018.

[152] V. Nagarajan and J. Kolter. Deterministic PAC-bayesian generalization bounds for deep networks via generalizing noise-resilience. In *International Conference on Learning Representations*, 2019.

[153] V. Nagarajan and J. Kolter. Generalization in deep networks: The role of distance from initialization. *arXiv preprint arXiv:1901.01672*, 2019.

[154] V. Nair and G. E. Hinton. Rectified linear units improve restricted Boltzmann machines. In *International Conference on Machine Learning*, pages 807–814, 2010.

[155] S. Nene, S. Nayar, and H. Murase. Columbia object image library (coil-100). *Citeseer*, 1996.

[156] S. Nene, S. Nayar, H. Murase, et al. Columbia object image library (coil-20). *Technical report CUCS-005-96*, 1996.

[157] G. Neofotistos, M. Mattheakis, G. D. Barmparis, J. Hizanidis, G. P. Tsironis, and E. Kaxiras. Machine learning with observers predicts complex spatiotemporal behavior. *arXiv preprint arXiv:1807.10758*, 2018.

[158] Y. Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

[159] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Ng. Reading digits in natural images with unsupervised feature learning. In *Advances in Neural Information Processing Systems*, 2011.

[160] B. Neyshabur, S. Bhojanapalli, D. McAllester, and N. Srebro. Exploring generalization in deep learning. In *Advances in Neural Information Processing Systems*, pages 5947–5956, 2017.

[161] B. Neyshabur, S. Bhojanapalli, and N. Srebro. A PAC-Bayesian approach to spectrally-normalized margin bounds for neural networks. *arXiv preprint arXiv:1707.09564*, 2017.

[162] B. Neyshabur, Z. Li, S. Bhojanapalli, Y. LeCun, and N. Srebro. The role of over-parametrization in generalization of neural networks. In *International Conference on Learning Representations*, 2019.

[163] B. Neyshabur, R. Salakhutdinov, and N. Srebro. Path-SGD: Path-normalized optimization in deep neural networks. In *Advances in Neural Information Processing Systems*, pages 2422–2430, 2015.

[164] B. Neyshabur, R. Tomioka, and N. Srebro. In search of the real inductive bias: On the role of implicit regularization in deep learning. *arXiv preprint arXiv:1412.6614*, 2014.

[165] A. Y. Ng. Feature selection, L_1 vs. L_2 regularization, and rotational invariance. In *International Conference on Machine Learning*, page 78, 2004.

[166] W. Ni, Y.-T. Cheng, C.-M. Cheng, and D. S. Grummon. An energy-based method for analyzing instrumented spherical indentation experiments. *Journal of Materials Research*, 19(1):149–157, 2004.

[167] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer, 2006.

[168] W. C. Oliver and G. M. Pharr. An improved technique for determining hardness and elastic modulus using load and displacement sensing indentation experiments. *Journal of Materials Research*, 7(6):1564–1583, 1992.

[169] W. C. Oliver and G. M. Pharr. Measurement of hardness and elastic modulus by instrumented indentation: Advances in understanding and refinements to methodology. *Journal of Materials Research*, 19(1):3–20, 2004.

[170] G. Pang, L. Lu, and G. E. Karniadakis. fPINNs: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

[171] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in PyTorch. 2017.

[172] J. C. Patra, R. N. Pal, B. Chatterji, and G. Panda. Identification of nonlinear dynamic systems using functional link artificial neural networks. *IEEE Transactions on Systems, Man, and Cybernetics, part B (Cybernetics)*, 29(2):254–262, 1999.

[173] A. Pinkus. Approximation theory of the MLP model in neural networks. *Acta Numerica*, 8:143–195, 1999.

[174] I. Podlubny. *Fractional differential equations: an introduction to fractional derivatives, fractional differential equations, to methods of their solution and some of their applications*. Elsevier, 1998.

[175] T. Poggio, K. Kawaguchi, Q. Liao, B. Miranda, L. Rosasco, X. Boix, J. Hidary, and H. Mhaskar. Theory of deep learning III: explaining the non-overfitting puzzle. *arXiv preprint arXiv:1801.00173*, 2017.

[176] T. Poggio, H. Mhaskar, L. Rosasco, B. Miranda, and Q. Liao. Why and when can deep-but not shallow-networks avoid the curse of dimensionality: a review. *International Journal of Automation and Computing*, 14(5):503–519, 2017.

[177] B. Poole, S. Lahiri, M. Raghu, J. Sohl-Dickstein, and S. Ganguli. Exponential expressivity in deep neural networks through transient chaos. In *Advances in Neural Information Processing Systems*, pages 3360–3368, 2016.

[178] T. Qin, K. Wu, and D. Xiu. Data driven governing equations approximation using deep neural networks. *Journal of Computational Physics*, 2019.

[179] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville. On the spectral bias of neural networks. In *International Conference on Machine Learning*, pages 5301–5310, 2019.

[180] M. Raissi, P. Perdikaris, and G. E. Karniadakis. Multistep neural networks for data-driven discovery of nonlinear dynamical systems. *arXiv preprint arXiv:1801.01236*, 2018.

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

[182] M. Raissi, A. Yazdani, and G. E. Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 2020.

[183] P. Ramachandran, B. Zoph, and Q. V. Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[184] F. Rossi and B. Conan-Guez. Functional multi-layer perceptron: A non-linear tool for functional data analysis. *Neural Networks*, 18(1):45–60, 2005.

[185] S. H. Rudy, S. L. Brunton, J. L. Proctor, and J. N. Kutz. Data-driven discovery of partial differential equations. *Science Advances*, 3(4):e1602614, 2017.

[186] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[187] I. Safran and O. Shamir. Spurious local minima are common in two-layer ReLU neural networks. In *International Conference on Machine Learning*, pages 4430–4438, 2018.

[188] A. Saigal, A. Giannakopoulos, H. Pettermann, and S. Suresh. Electrical response during indentation of a 1-3 piezoelectric ceramic-polymer composite. *Journal of Applied Physics*, 86(1):603–606, 1999.

[189] T. Salimans and D. P. Kingma. Weight normalization: A simple reparameterization to accelerate training of deep neural networks. In *Advances in Neural Information Processing Systems*, pages 901–909, 2016.

[190] B. Sanchez-Lengeling and A. Aspuru-Guzik. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018.

[191] A. M. Saxe, Y. Bansal, J. Dapello, M. Advani, A. Kolchinsky, B. D. Tracey, and D. D. Cox. On the information bottleneck theory of deep learning. *Journal of Statistical Mechanics: Theory and Experiment*, 2019(12):124020, 2019.

[192] A. M. Saxe, J. L. McClelland, and S. Ganguli. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. In *International Conference on Learning Representations*, 2014.

[193] Z. Shi, E. Tsymbalov, M. Dao, S. Suresh, A. Shapeev, and J. Li. Deep elastic strain engineering of bandgap through machine learning. *Proceedings of the National Academy of Sciences*, 116(10):4117–4122, 2019.

[194] R. Shwartz-Ziv and N. Tishby. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810*, 2017.

[195] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484, 2016.

[196] J. Sirignano and K. Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018.

[197] J. Sokolic, R. Giryes, G. Sapiro, and M. Rodrigues. Generalization error of invariant classifiers. *arXiv preprint arXiv:1610.04574*, 2016.

[198] J. Sokolic, R. Giryes, G. Sapiro, and M. Rodrigues. Robust large margin deep neural networks. *IEEE Transactions on Signal Processing*, 65(16):4265–4280, 2017.

[199] D. Soudry, E. Hoffer, M. Nacson, S. Gunasekar, and N. Srebro. The implicit bias of gradient descent on separable data. *Journal of Machine Learning Research*, 19(1):2822–2878, 2018.

[200] B. F. Spencer Jr, V. Hoskere, and Y. Narazaki. Advances in computer vision-based civil infrastructure inspection and monitoring. *Engineering*, 2019.

[201] S. Sridhar, A. Giannakopoulos, S. Suresh, and U. Ramamurty. Electrical response during indentation of piezoelectric materials: A new method for material characterization. *Journal of Applied Physics*, 85(1):380–387, 1999.

[202] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[203] R. K. Srivastava, K. Greff, and J. Schmidhuber. Training very deep networks. In *Advances in Neural Information Processing Systems*, pages 2377–2385, 2015.

[204] Y. Sun, X. Wang, and X. Tang. Deeply learned face representations are sparse, selective, and robust. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2892–2900, 2015.

[205] Z.-z. Sun and X. Wu. A fully discrete difference scheme for a diffusion-wave system. *Applied Numerical Mathematics*, 56(2):193–209, 2006.

[206] S. Suresh. Graded materials for resistance to contact deformation and damage. *Science*, 292(5526):2447–2451, 2001.

[207] S. Suresh and A. Giannakopoulos. A new method for estimating residual stresses by instrumented sharp indentation. *Acta Materialia*, 46(16):5755–5767, 1998.

[208] I. Sutskever, J. Martens, G. Dahl, and G. Hinton. On the importance of initialization and momentum in deep learning. In *International Conference on Machine Learning*, pages 1139–1147, 2013.

[209] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[210] D. Tabor. The hardness of metals, 1951.

[211] K. S. Tai, P. Bailis, and G. Valiant. Equivariant transformer networks. *arXiv preprint arXiv:1901.11399*, 2019.

[212] A. M. Tartakovsky, C. O. Marrero, P. Perdikaris, G. D. Tartakovsky, and D. Barajas-Solano. Learning parameters and constitutive relationships with physics informed deep neural networks. *arXiv preprint arXiv:1808.03398*, 2018.

[213] E. Teller. *The pursuit of simplicity*. Pepperdine University Press, 1981.

[214] N. Trask, R. G. Patel, B. J. Gross, and P. J. Atzberger. GMLS-Nets: A framework for learning from unstructured data. *arXiv preprint arXiv:1909.05371*, 2019.

[215] L. Trottier, P. Gigu, B. Chaib-draa, et al. Parametric exponential linear unit for deep convolutional neural networks. In *2017 16th IEEE International Conference on Machine Learning and Applications (ICMLA)*, pages 207–214. IEEE, 2017.

[216] E. Tyulyukovskiy and N. Huber. Identification of viscoplastic material parameters from spherical indentation data: Part I. neural networks. *Journal of Materials Research*, 21(3):664–676, 2006.

[217] D. Ulyanov, A. Vedaldi, and V. Lempitsky. Instance normalization: The missing ingredient for fast stylization. *arXiv preprint arXiv:1607.08022*, 2016.

[218] B. P. van Milligen, V. Tribaldos, and J. Jiménez. Neural network differential equation and plasma equilibrium solver. *Physical Review Letters*, 75(20):3594, 1995.

[219] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017.

[220] L. Wang, M. Ganor, and S. Rokhlin. Inverse scaling functions in nanoindentation with sharp indenters: Determination of material properties. *Journal of Materials Research*, 20(4):987–1001, 2005.

[221] C. Wei and T. Ma. Data-dependent sample complexity of deep neural networks via Lipschitz augmentation. *arXiv preprint arXiv:1905.03684*, 2019.

[222] B. J. West. *Fractional Calculus View of Complexity: Tomorrow's Science*. CRC Press, 2016.

[223] N. Winovich, K. Ramani, and G. Lin. ConvPDE-UQ: Convolutional neural networks with quantified uncertainty for heterogeneous elliptic partial differential equations on varied domains. *Journal of Computational Physics*, 394:263–279, 2019.

[224] C. Wu, J. Luo, and J. Lee. No spurious local minima in a two hidden unit ReLU network. In *International Conference on Learning Representations Workshop*, 2018.

[225] Y. Wu and K. He. Group normalization. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 3–19, 2018.

[226] Z. Xu, Y. Zhang, T. Luo, Y. Xiao, and Z. Ma. Frequency principle: Fourier analysis sheds light on deep neural networks. *arXiv preprint arXiv:1901.06523*, 2019.

[227] L. Yang, D. Zhang, and G. E. Karniadakis. Physics-informed generative adversarial networks for stochastic differential equations. *arXiv preprint arXiv:1811.02033*, 2018.

[228] T. Young, D. Hazarika, S. Poria, and E. Cambria. Recent trends in deep learning based natural language processing. *IEEE Computational Intelligence Magazine*, 13(3):55–75, 2018.

[229] C. Yun, S. Sra, and J. A. Small nonlinearities in activation functions create bad local minima in neural networks. *arXiv preprint arXiv:1802.03487*, 2018.

[230] M. Zayernouri and G. E. Karniadakis. Fractional Sturm–Liouville eigen-problems: theory and numerical approximation. *Journal of Computational Physics*, 252:495–517, 2013.

[231] M. D. Zeiler. ADADELTA: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[232] C. Zhang, S. Bengio, M. Hardt, B. Recht, and O. Vinyals. Understanding deep learning requires rethinking generalization. *arXiv preprint arXiv:1611.03530*, 2016.

[233] C. Zhang, Q. Liao, A. Rakhlin, B. Miranda, N. Golowich, and T. Poggio. Theory of deep learning IIb: Optimization properties of SGD. *arXiv preprint arXiv:1801.02254*, 2018.

[234] D. Zhang, L. Guo, and G. E. Karniadakis. Learning in modal space: Solving time-dependent stochastic PDEs using physics-informed neural networks. *arXiv preprint arXiv:1905.01205*, 2019.

[235] D. Zhang, L. Lu, L. Guo, and G. E. Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850, 2019.

[236] P. Zhang, S. Li, and Z. Zhang. General relationship between strength and hardness. *Materials Science and Engineering: A*, 529:62–73, 2011.

[237] Y. Zhang, J. D. Hart, and A. Needleman. Identification of plastic properties from conical indentation using a Bayesian-type statistical approach. *Journal of Applied Mechanics*, 86(1), 2019.

[238] Z. Zhang and G. E. Karniadakis. *Numerical methods for stochastic partial differential equations with white noise*. Springer, 2017.

[239] Z. Zhang, F. Zeng, and G. E. Karniadakis. Optimal error estimates of spectral Petrov-Galerkin and collocation methods for initial value problems of fractional differential equations. *SIAM Journal on Numerical Analysis*, 53(4):2074–2096, 2015.

[240] H. Zhao and J. Zhang. Nonlinear dynamic system identification using pipelined functional link artificial recurrent neural network. *Neurocomputing*, 72(13-15):3046–3054, 2009.

[241] W. Zhou, V. Veitch, M. Austern, R. Adams, and P. Orbanz. Compressibility and generalization in large-scale deep learning. *arXiv preprint arXiv:1804.05862*, 2018.

[242] Y. Zhou and Y. Liang. Critical points of neural networks: Analytical forms and landscape properties. *arXiv preprint arXiv:1710.11205*, 2017.

[243] Y. Zhu, N. Zabaras, P.-S. Koutsourelakis, and P. Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

[244] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.