SOLVING PDES ON UNKNOWN MANIFOLDS WITH MACHINE LEARNING

A PREPRINT

Senwei Liang
Department of Mathematics, Purdue University, IN 47907, USA
liang339@purdue.edu

Shixiao W. Jiang
Institute of Mathematical Sciences, ShanghaiTech University, Shanghai, 201210, China
jiangshx@shanghaitech.edu.cn

John Harlim
Department of Mathematics, Department of Meteorology and Atmospheric Science,
Institute for Computational and Data Sciences
The Pennsylvania State University, University Park, PA 16802, USA
jharlim@psu.edu

Haizhao Yang
Department of Mathematics, University of Maryland, College Park, MD 20742, USA
hzyang@umd.edu

February 28, 2024

ABSTRACT

This paper proposes a mesh-free computational framework and machine learning theory for solving elliptic PDEs on unknown manifolds, identified with point clouds, based on diffusion maps (DM) and deep learning. The PDE solver is formulated as a supervised learning task to solve a least-squares regression problem that imposes an algebraic equation approximating a PDE (and boundary conditions if applicable). This algebraic equation involves a graph-Laplacian type matrix obtained via DM asymptotic expansion, which is a consistent estimator of second-order elliptic differential operators. The resulting numerical method is to solve a highly non-convex empirical risk minimization problem subjected to a solution from a hypothesis space of neural networks (NNs). In a well-posed elliptic PDE setting, when the hypothesis space consists of neural networks with either infinite width or depth, we show that the global minimizer of the empirical loss function is a consistent solution in the limit of large training data. When the hypothesis space is a two-layer neural network, we show that for a sufficiently large width, gradient descent can identify a global minimizer of the empirical loss function. Supporting numerical examples demonstrate the convergence of the solutions, ranging from simple manifolds with low and high co-dimensions, to rough surfaces with and without boundaries. We also show that the proposed NN solver can robustly generalize the PDE solution on new data points with generalization errors that are almost identical to the training errors, superseding a Nyström-based interpolation method.

***K*eywords** High-Dimensional PDEs • Diffusion Maps • Deep Neural Networks • Convergence Analysis • Least-Squares Minimization •Manifolds •Point Clouds.

1 Introduction

Solving high-dimensional PDEs on unknown manifolds is a challenging computational problem that commands a wide variety of applications. In physics and biology, such a problem arises in modeling of granular flow [80], liquid crystal [92], biomembranes [31]. In computer graphics [11], PDEs on surfaces have been used to restore damaged patterns on a surface [67], brain imaging [70], among other applications. By unknown manifolds, we refer to the situation where the parameterization of the domain is unknown. The main computational challenge arising from this constraint is on the approximation of the differential operator using the available sample data (point clouds) that are assumed to lie on (or close to) a smooth manifold. Among many available methods proposed for PDEs on surfaces embedded in $\mathbf{R}^3$, they typically parameterize the surface and subsequently use it to approximate the tangential derivatives along surfaces. For example, the finite element method represents surfaces [26, 13, 12] using triangular meshes. Thus its accuracy relies on the quality of the generated meshes which may be poor if the given point cloud data are randomly distributed. Another class of approach is to estimate the embedding function of the surface using e.g., level set representation [11] or closest point representation [81, 14, 68], and subsequently solve the embedded PDE on the ambient space. The key issue with this class of approaches is that since the embedded PDE is at least one dimension higher than the dimension of the two-dimensional surface (i.e., co-dimension higher than one), the computational cost may not be feasible if the manifold is embedded in high-dimensional ambient space. Another class of approaches is the mesh-free radial basis function (RBF) method [78, 34] for solving PDEs on surfaces. This approach, however, may not be robust in high dimensional and rough surface problems as pointed out in [34, 83], and the convergence near the boundary can be problematic.

Motivated by [57], an unsupervised learning method called the *Diffusion Map* (DM) algorithm [16] was proposed to directly solve the second-order elliptic PDE on point clouds data that lie on the manifolds [36]. The proposed DM-based solver has been extended to elliptic problems with various types of boundary conditions that typically arise in applications [51], such as non-homogeneous Dirichlet, Neumann, and Robin types and to time-dependent advection-diffusion PDEs [94]. The main advantage of this approach is to avoid the tedious parameterization of sub-manifolds in a high-dimensional space. However, the estimated solution is represented by a discrete vector whose components approximate the function values on the available point clouds, analogous to standard finite-difference methods. With such a representation, one will need an interpolation method to find the solutions on new data points, which is a nontrivial task when the domain is an unknown manifold. This issue is particularly relevant if the available data points come sequentially. Another problem with the DM-based solver is that the size of the matrix approximating the differential operator increases as a function of the data size, which creates a computational bottleneck when the PDE problem involves solving a singular linear system with a pseudo-inversion or an eigenvalue decomposition.

One way to overcome these computational issues is to solve the PDE in a supervised learning framework using neural networks (NNs). On an Euclidean domain, where extensive research has been conducted [27, 41, 52, 91, 7, 100, 56, 6, 79, 33], NN-based PDE solvers reformulate a PDE problem as a regression problem. Subsequently, the PDE solution is approximated by a class of NN functions. NNs have good approximation properties [5, 28, 29, 74, 89, 19, 73, 48, 49, 85, 86, 20] that enable application of these PDE solvers to high-dimensional problems. With the advanced computational tools (e.g., TensorFlow and Pytorch) and the built-in optimization algorithms therein, developing mathematical software with parallel computing using NNs is much simpler than conventional numerical techniques. In fact, NN has been proposed to solve linear problems [15, 66, 38].

Building upon this encouraging result, we propose to solve PDEs on unknown manifolds by embedding the DM algorithm in NN-based PDE solvers. In particular, the DM algorithm is employed to approximate the second-order elliptic differential operator defined on the manifolds. Subsequently, a least-squares regression problem is formulated by imposing an algebraic equation that involves a graph-Laplacian type matrix, obtained by the discretization of the DM asymptotic expansion on the available point cloud training data. Numerically, we solve the resulting empirical loss function by finding the solution from a hypothesis space of neural-network type functions (e.g., with feedforward NNs, we consider the compositions of power of ReLU or polynomial-sine activation functions). The proposed NN approach provides a more feasible approach to solve large linear systems associated with manifold PDEs compared to performing (a stable) pseudo-inversion and obtain a continuous function solution instead of a discrete solution at training points. Such feasibility could be achieved through mini-batch training as we shall demonstrate in Section 5.2.2, which circumvents the computational and memory issues of using the full linear system at once. Similar mini-batch training strategies have also been used in [38]. Theoretically, we study the approximation and optimization aspects of the error analysis, induced by the training procedure that minimizes the empirical risk defined on available point cloud data. Under appropriate regularity assumptions, we show that when the hypothesis space has either an infinite width or depth, the global minimizer of the empirical loss function is a consistent solution in the limit of large training data. The corresponding error bound gives the

relations between the desired accuracy and the required size of training data and width (or depth) of the network. Furthermore, when the hypothesis space is a two-layer NN, we show that for a sufficiently large width, the gradient descent (GD) method can identify a global minimizer of the empirical loss function. Numerically, we verify the proposed methods on several test problems on simple 2D and 3D manifolds of various co-dimensions and rough unknown surfaces with and without boundaries. In a set of instructive examples, we compare the accuracy of the generalization of the NN solution on new data points to the interpolation solution attained by applying the classical Nyström scheme on a linear combination of the estimated Laplace-Beltrami eigenfunctions that spans the PDE solution. In rough surface examples, we will verify the accuracy of the solutions by comparing them against the Finite Element Method solutions.

The paper will be organized as follows. In Section 2, we give a brief review on the DM (and variable bandwidth DM) based PDE solver on closed manifolds and an overview of the ghost point diffusion maps (GPDM) for manifolds with boundaries. The proposed PDE solver is introduced in Section 3. The theoretical foundation of the proposed method is presented in Section 4. The numerical performance of the proposed method is illustrated in Section 5. We conclude the paper with a summary in Section 6. For reader's convenience, we present an algorithmic perspective for GPDM in Appendix A. We also include the longer proof for the optimization aspect of the algorithm in Appendix B and report all parameters used to generate the numerical results in Appendix C.

2 DM-based PDE Solver on Unknown Manifolds

To illustrate the main idea, let us discuss an elliptic problem defined on a d-dimensional closed sub-manifold $M \subseteq [0,1]^n \subseteq \mathbf{R}^n$. We will provide a brief discussion for the problem defined on compact manifolds with boundaries to end this section. Let $u : M \to \mathbf{R}$ be a solution of the elliptic PDE,

$$(-a(\mathbf{x}) + L)u(\mathbf{x}) := -a(\mathbf{x})u(\mathbf{x}) + \mathrm{div}_g\left(\kappa(\mathbf{x})\nabla_g u(\mathbf{x})\right) = f(\mathbf{x}), \quad \mathbf{x} \in M \tag{1}$$

Here, we have used the notations div_g and ∇_g for the divergence and gradient operators, respectively, defined with respect to the Riemannian metric g inherited by M from the ambient space $\mathbf{R}^n$. The real-valued functions a and κ are strictly positive such that $(-a + L)$ is strictly negative definite. The problem is assumed to be well-posed for $f \in C^{1,\alpha}(M)$, for $0 < \alpha < 1/2$. For $a \in C^{1,\alpha}(M)$ and $\kappa \in C^{3,\alpha}(M)$, a unique classical solution $u \in C^{3,\alpha}(M)$ is guaranteed. Here, we raise the regularity by one-order of derivative (compared to reported results in the literature [43, 37]) for the following reason.

The key idea of the DM-based PDE solver rests on the following asymptotic expansion [16],

$$G_\epsilon u(\mathbf{x}) := \epsilon^{-d/2} \int_M h\left(\frac{\|\mathbf{x}-\mathbf{y}\|^2}{\epsilon}\right) u(\mathbf{y}) dV(\mathbf{y}) = u(\mathbf{x}) + \epsilon(\omega(\mathbf{x})u(\mathbf{x}) + \Delta_g u(\mathbf{x})) + O(\epsilon^2), \tag{2}$$

where the second equality is valid for any $u \in C^3(M)$ and any $\mathbf{x} \in M$. To employ this asymptotic expansion in our approximation, we raise the regularity of the classical solution to be C^3 instead of the usual regularity assumption where $u \in C^2$ for continuous f. Here, the function $h : [0,\infty) \to (0,\infty)$ is defined as $h(s) = \frac{e^{-s/4}}{(4\pi)^{d/2}}$ such that, effectively for a fixed bandwidth parameter $\epsilon > 0$, G_ϵ is a local integral operator. In (2), V denotes the volume form inherited by the manifold from the ambient space, the term ω depends on the geometry, $\Delta_g = \mathrm{div}_g \nabla_g$ denotes the negative-definite Laplace-Beltrami operator, $\|\bullet\|$ denotes the standard Euclidean norm for vectors in $\mathbf{R}^n$ and we will use the same notation for arbitrary finite-dimensional vector space. Based on the asymptotic expansion in (2), one can approximate the differential operator L as follows,

$$L u(\mathbf{x}) = \frac{\kappa(\mathbf{x})}{\epsilon}\left(G_\epsilon(u(\mathbf{x})\sqrt{\kappa(\mathbf{x})}) - u(\mathbf{x})G_\epsilon\sqrt{\kappa(\mathbf{x})}\right) + O(\epsilon) := L_\epsilon u(\mathbf{x}) + O(\epsilon). \tag{3}$$

In our setup, we assume that we are given a set of point cloud data $X := \{\mathbf{x}_i \in M\}_{i=1,\dots,N}$, independent and identically distributed (i.i.d.) according to π, with an empirical measure defined as, $\pi_N(\mathbf{x}) = \frac{1}{N}\sum_{i=1}^N \delta_{\mathbf{x}_i}(\mathbf{x})$. We use the notation $L^2(\pi)$ to denote the space of square-integrable functions with respect to the measure π. Accordingly, we define $L^2(\pi_N)$ as the space of functions $u : X \to \mathbf{R}^n$, endowed with the inner-product and norm-squared defined as,

$$\langle u, u\rangle_{L^2(\pi_N)} = \|u\|^2_{L^2(\pi_N)} = \int_M u^2(x) d\pi_N(x) = \frac{1}{N}\sum_{i=1}^N u^2(\mathbf{x}_i).$$

Given point cloud data, we first approximate the sampling density, $q = d\pi/dV$ evaluated at $\mathbf{x}_i$, with $Q_i := \epsilon^{-d/2} N^{-1} \sum_{j=1}^N h\left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\epsilon}\right)$. Define $\mathbf{W} \in \mathbf{R}^{N\times N}$ with entries,

$$\mathbf{W}_{ij} := \epsilon^{-d/2-1} N^{-1} h\left(\frac{\|\mathbf{x}_i - \mathbf{x}_j\|^2}{\epsilon}\right) \kappa(\mathbf{x}_i)\kappa(\mathbf{x}_j) Q_i^{-1} Q_j^{-1} \tag{4}$$

Define also a diagonal matrix $\mathbf{D} \in \mathbf{R}^{N\times N}$ with diagonal entries, $\mathbf{D}_{ii} = \sum_{j=1}^{N} \mathbf{W}_{ij}$. Then, we approximate the integral operator in (3) with the matrix $\mathbf{L}_\epsilon := \mathbf{W}-\mathbf{D}$, similarly to a discrete unnormalized graph Laplacian matrix. Here, the matrix $\mathbf{L}_\epsilon$ is self-adjoint and semi negative-definite with respect to the inner-product in $\langle u, v\rangle_Q := \frac{1}{N}\sum_{i=1}^{N} u(\mathbf{x}_i)v(\mathbf{x}_i)Q_i^{-1}$ such that it admits a non-positive spectrum, $0 = \lambda_1 > \lambda_2 \geq \ldots \geq \lambda_N$ with eigenvectors orthonormal in $\langle\bullet,\bullet\rangle_Q$. Since the kernel function h decays exponentially, the k-nearest-neighbor algorithm is usually used to impose sparsity to the estimator $\mathbf{L}_\epsilon$. The DM-based PDE solver approximates the PDE solution $u(\mathbf{x}_j)$ using the i-th component of the vector $\mathbf{u}_\epsilon \in \mathbf{R}^N$ that satisfies the linear system

$$(-\mathbf{a}+\mathbf{L}_\epsilon)\mathbf{u}_\epsilon = \mathbf{f} \tag{5}$$

of size N. Here, the i-th diagonal component of the diagonal matrix $\mathbf{a} \in \mathbf{R}^{N\times N}$ and the i-th component of $\mathbf{f} \in \mathbf{R}^N$ are $a(\mathbf{x}_i)$ and $f(\mathbf{x}_i)$, respectively. In [36], this approach has been theoretically justified and numerically extended to approximate the non-symmetric, uniformly elliptic second-order differential operators associated to the generator of Itô diffusions with appropriate local kernel functions.

When the sampling density is not bounded away from zero or too far away from uniform distribution, the above operator estimation obtained from the fixed bandwidth kernels can be unbounded. This issue can be overcome by applying variable bandwidth kernels of the form:

$$K_{\epsilon,\rho}(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{4\epsilon\rho(\mathbf{x})\rho(\mathbf{y})}\right), \tag{6}$$

where ρ is a bandwidth function, chosen to be inversely proportional to the sampling density q. Since the sampling density is usually unknown, we first need to approximate it. While there are many ways to estimate density, in our algorithm we employ kernel density estimation with the following kernel that is closely related to the cKNN [10] and self-tuning kernel [101],

$$K_{\epsilon,0}(\mathbf{x},\mathbf{y}) = \exp\left(-\frac{\|\mathbf{x}-\mathbf{y}\|^2}{2\epsilon\rho_0(\mathbf{x})\rho_0(\mathbf{y})}\right),$$

where $\rho_0(\mathbf{x}) := \left(\frac{1}{k_2-1}\sum_{j=2}^{k_2}\|\mathbf{x}-\mathbf{x}_j\|^2\right)^{1/2}$ denotes the average distance of $\mathbf{x}$ to the first k_2-nearest neighbors $\{\mathbf{x}_j\}$, $j = 2,\ldots k_2$ excluding itself. Using this kernel, the sampling density $q(\mathbf{x})$ is estimated by $Q(\mathbf{x}) = \sum_{j=1}^{N} K_{\epsilon,0}(\mathbf{x},\mathbf{x}_j)/\rho_0(\mathbf{x})^d$ at given point cloud data.

Before we estimate $\mathcal{L}$ with the variable bandwidth kernel in (6), let us give a brief overview of the estimation of the Laplace-Beltrami operator that occurs in the asymptotic expansion of the integral operator defined with kernel $K_{\epsilon,\rho}$. We refer interested readers to Eq.(A.12) in [9]) for the detailed of the asymptotic expansion, which is a generalization of (2), as it involves variable bandwidth function ρ. To estimate the Laplace-Beltrami operator, one chooses the bandwidth function to be $\rho(\mathbf{x}) = q(\mathbf{x})^\beta \approx Q(\mathbf{x})^\beta$, with $\beta = -1/2$. With this bandwidth function, we employ the DM algebraic steps to approximate the Laplace-Beltrami operator. That is, we define $Q_\rho(\mathbf{x}) = \sum_{j=1}^{N} K_{\epsilon,\rho}(\mathbf{x},\mathbf{x}_j)/\rho(\mathbf{x})^d$. Then, we remove the sampling bias by applying a right normalization $K_{\epsilon,\rho,\alpha}(\mathbf{x}_i,\mathbf{x}_j) = \frac{K_{\epsilon,\rho}(\mathbf{x}_i,\mathbf{x}_j)}{Q_\rho(\mathbf{x}_i)^\alpha Q_\rho(\mathbf{x}_j)^\alpha}$, where $\alpha = -d/4+1/2$. We refer to [44, 9] for more details about the variable bandwidth diffusion map (VBDM) estimator of weighted Laplacian with other choices of α and β. Define diagonal matrices $\mathbf{Q}$ and $\mathbf{P}$ with entries $\mathbf{Q}_{ii} = Q_\rho(\mathbf{x}_i)$ and $\mathbf{P}_{ii} = \rho(\mathbf{x}_i)$, respectively, and also define the symmetric matrix $\mathbf{K}$ with entries $\mathbf{K}_{ij} = K_{\epsilon,\rho,\alpha}(\mathbf{x}_i,\mathbf{x}_j)$. Next, one can obtain the VBDM estimator, $\mathbf{L}_{\epsilon,\rho} := \mathbf{P}^{-2}(\mathbf{D}^{-1}\mathbf{K}-\mathbf{I})/\epsilon$, where $\mathbf{I}$ is an identity matrix, as a discrete estimator to the Laplace-Beltrami operator in high probability.

To approximate the differential operator $\mathcal{L}$ in (1), we use the fact that,

$$\mathrm{div}_g(\kappa(\mathbf{x})\nabla_g u(\mathbf{x})) = \frac{\pi}{\kappa}\left[\Delta_g\left(u\frac{\pi}{\kappa}\right) - u\Delta_g\frac{\pi}{\kappa}\right].$$

This relation suggests that we can estimate $\mathcal{L}$ in (1) with the matrix $\mathbf{L}^\kappa_{\epsilon,\rho} := \mathbf{A}\mathbf{L}_{\epsilon,\rho}\mathbf{A}-\mathbf{E}$, where $\mathbf{A}$ and $\mathbf{E}$ are diagonal matrices with entries $\mathbf{A}_{ii} = \frac{\pi}{\kappa}(\mathbf{x}_i)$ and $\mathbf{E}_{ii} = (\mathbf{A}\mathbf{L}_{\epsilon,\rho}\mathbf{A}\mathbf{1})_i$ with $\mathbf{1}$ being a vector with all entries equal to 1. Here, the property of $\mathbf{L}^\kappa_{\epsilon,\rho}$ is similar to that of a discrete estimator $\mathbf{L}_{\epsilon,\rho}$. The matrix $\mathbf{L}^\kappa_{\epsilon,\rho}$ is self-adjoint and semi negative definite with respect to the inner-product in $\langle u, v\rangle_{\mathbf{S}} := \frac{1}{N}\sum_{i=1}^{N} u(\mathbf{x}_i)v(\mathbf{x}_i)\mathbf{S}_{ii}^2$, where $\mathbf{S}_{ii}$ is the diagonal entries of $\mathbf{S} = \mathbf{P}\mathbf{D}^{1/2}$. Then it is clear that $\mathbf{L}^\kappa_{\epsilon,\rho}$ also admits a non-positive spectrum $\{\lambda_i\}_{i=1}^N$ with $\lambda_1 = 0$ and an associated basis of eigenvectors $\{\psi_i\}_{i=1}^N$ orthonormal in $\langle\bullet,\bullet\rangle_{\mathbf{S}}$ with $\psi_1 \equiv 1$.

Thus, we have two estimators, $\mathbf{L}_\epsilon$ obtained via the fixed bandwidth kernel h in (2) and $\mathbf{L}^\kappa_{\epsilon,\rho}$ obtained via the variable bandwidth kernel in (6), for the differential operator $\mathcal{L}$. As we noted before, the motivation for VBDM estimation is to overcome samples that are far away from uniform. While theoretically, they induce the same estimate, practically,

we found the VBDM estimate is more robust to the choice of ϵ. Particularly, it produces accurate estimation when we specify ϵ using the auto-tuning method proposed in [17] that is practically more convenient than hand-tuning ϵ. To summarize, we refer to the VBDM-based PDE solver as the linear system in (5) but with DM estimator $\mathbf{L}_\epsilon$ replaced with the VBDM estimator $\mathbf{L}^{\kappa}_{\epsilon,\rho}$. In the remainder of this paper, we will only use the notation $\mathbf{L}_\epsilon$ as a discrete estimator of $\mathcal{L}$ and understand it as $\mathbf{L}^{\kappa}_{\epsilon,\rho}$ when VBDM is used.

Beyond the no boundary case, the approximation in (3), unfortunately, will not produce an accurate approximation when $\mathbf{x}$ is sufficiently closed to the boundary. To overcome this issue, a modified DM algorithm, following the classical ghost points method to obtain a higher-order finite-difference approximation of Neumann problems (e.g. [55]) was proposed in [51]. The proposed method, which is called the *Ghost Point Diffusion Maps* (GPDM), appends the point clouds with a set of ghost points away from the boundary along the outward normal collar such that the resulting discrete estimator is consistent in the $L^2(\mu_N)$-sense, averaged over N interior points on the manifold M [94]. To simplify the discussion in the remainder of this paper, we will use the same notation $\mathbf{L}_\epsilon$ to denote the discrete estimate of $\mathcal{L}$, obtained either via the classical or the ghost points diffusion maps. In Appendix A, we provide a brief review on the GPDM for Dirichlet boundary value problems.

3 Solving PDEs on Unknown Manifolds using Diffusion Maps and Neural Networks

We now present a new hybrid algorithm based on DM and NNs.

3.1 Deep neural networks (DNNs)

Mathematically, DNNs are highly nonlinear functions constructed by compositions of simple nonlinear functions. For simplicity, we consider the fully connected feed-forward neural network (FNN), which is the composition of L simple nonlinear functions as follows: $\varphi(\mathbf{x};\boldsymbol{\vartheta}) := \mathbf{a}\cdot\mathbf{h}_{L-1}\circ\mathbf{h}_{L-2}\circ\cdots\circ\mathbf{h}_1(\mathbf{x})$, where $\mathbf{h}_\ell(\mathbf{x}) = \sigma(\mathbf{W}_\ell\mathbf{x}+\mathbf{b}_\ell)$ with $\mathbf{W}_\ell \in \mathbf{R}^{N_\ell\times N_{\ell-1}}$, $\mathbf{b}_\ell \in \mathbf{R}^{N_\ell}$ for $\ell = 1,\ldots,L$, $\mathbf{a}\in\mathbf{R}^{N_L}$, σ is a nonlinear activation function, e.g., a rectified linear unit (ReLU) $\sigma(x) = \max\{x,0\}$. Each $\mathbf{h}_\ell$ is referred as a hidden layer, N_ℓ is the width of the ℓ-th layer, and L is called the depth of the FNN. In the above formulation, $\boldsymbol{\vartheta} := \{\mathbf{a}, \mathbf{W}_\ell, \mathbf{b}_\ell : 1\le\ell\le L\}$ denotes the set of all parameters in φ. For simplicity, we focus on FNN with a uniform width m, i.e., $N_\ell = m$ for all $\ell = 0$, in this paper.

3.2 Supervised Learning

Supervised learning approximates an unknown target function $f : \mathbf{x}\in\Omega\to y\in\mathbf{R}$ from training samples $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$, where $\mathbf{x}_i$'s are usually assumed to be i.i.d samples from an underlying distribution π defined on a domain $\Omega\subseteq\mathbf{R}^h$, and $y_i = f(\mathbf{x}_i)$. Consider the square loss $\ell(\mathbf{x},y;\boldsymbol{\vartheta}) = \frac{1}{2}\|\varphi(\mathbf{x};\boldsymbol{\vartheta})-y\|^2$ of a given DNN $\varphi(\mathbf{x};\boldsymbol{\vartheta})$ that is used to approximate $f(\mathbf{x})$, the population risk (error) and empirical risk (error) functions are, respectively,

$$J(\boldsymbol{\vartheta}) = \frac{1}{2}\mathbf{E}_{\mathbf{x}\sim\pi}\left[\|\varphi(\mathbf{x};\boldsymbol{\vartheta})-f(\mathbf{x})\|^2\right], \qquad \hat{J}(\boldsymbol{\vartheta}) = \frac{1}{2N}\sum_{i=1}^N\|\varphi(\mathbf{x}_i;\boldsymbol{\vartheta})-y_i\|^2. \tag{7}$$

The optimal set $\hat{\boldsymbol{\vartheta}}$ is identified via $\hat{\boldsymbol{\vartheta}} = \arg\min_{\boldsymbol{\vartheta}}\hat{J}(\boldsymbol{\vartheta})$, and $\varphi(\mathbf{x};\hat{\boldsymbol{\vartheta}})$ is the learned DNN that approximates the unknown function f.

3.3 The NN-based PDE solver with DM

Solving a PDE can be transformed into a supervised learning problem. Physical laws, like PDEs and boundary conditions, are used to generate training data in a supervised learning problem to infer the solution of PDEs. In the case when the PDE is defined on a manifold, we propose to use DM as an approximation to the differential operator according to (3) to obtain a linear system in (5), apply NN to parametrize the PDE solution, and adopt a least-square framework to identify the NN that approximates the PDE solution. For example, to solve the PDE problem in (1) in the strong sense on a d-dimensional closed, smooth manifold M identified with points $X = \{\mathbf{x}_1,\ldots,\mathbf{x}_N\}\subset M$, we minimize the following empirical loss

$$\boldsymbol{\vartheta}_S = \arg\min_{\boldsymbol{\vartheta}} J_{S,\epsilon}(\boldsymbol{\vartheta}) := \arg\min_{\boldsymbol{\vartheta}}\frac{1}{2}\|(\mathbf{a}+\mathbf{L}_\epsilon)\boldsymbol{\varphi}_{\boldsymbol{\vartheta}}-\mathbf{f}\|^2_{L^2(\pi_N)}, \tag{8}$$

where $\mathbf{L}_\epsilon\in\mathbf{R}^{N\times N}$ denotes the DM estimator for the differential operator $\mathcal{L}$, $\mathbf{a}$ and $\mathbf{f}$ are defined in (5), $\boldsymbol{\varphi}_{\boldsymbol{\vartheta}}\in\mathbf{R}^N$ with the i-th entry as $\varphi(\mathbf{x}_i;\boldsymbol{\vartheta})$. When $\mathbf{a} = \mathbf{0}$, we add a regularization term $\gamma\|\boldsymbol{\varphi}_{\boldsymbol{\vartheta}}\|^2_{L^2(\pi_N)}$ in the loss function (8) to guarantee well-posedness, where $\gamma > 0$ is a regularization parameter. When stochastic gradient descent (SGD) is used to

minimize (8), a small subset of the given point clouds is randomly selected in each iteration. This amounts to randomly choosing batches, consisting of a few rows of $(\mathbf{a}+\mathbf{L}_\epsilon)$ and a few entries of $\mathbf{f}$, to approximate the empirical loss function.

In the case of Dirichlet problems with non-homogeneous boundary conditions, $u(\mathbf{x}) = g(\mathbf{x})$, $\forall \mathbf{x} \in \partial M$, given boundary points $\{\mathbf{x}_1, \ldots, \mathbf{x}_{N_b}\} \subset X \cap \partial M$ as the last N_b points of X, a penalty term is added to (8) to enforce the boundary condition as follows:

$$\boldsymbol{\vartheta}_S := \arg\min_{\boldsymbol{\vartheta}} \frac{1}{2}\|(\mathbf{a}+\mathbf{L}_\epsilon)\boldsymbol{\varphi}_{\boldsymbol{\vartheta}} - \mathbf{f}\|^2_{L^2(\pi_{N-N_b})} + \frac{\lambda}{2}\|\boldsymbol{\varphi}^b_{\boldsymbol{\vartheta}} - \mathbf{g}\|^2_{L^2(\pi_{N_b})}, \tag{9}$$

where $\mathbf{L}_\epsilon \in \mathbf{R}^{(N-N_b)\times N}$ denotes the GPDM estimator for the differential operator L defined for problem with boundary and $\lambda > 0$ is a hyper-parameter. The construction of the matrix $\mathbf{L}_\epsilon$ is discussed in Appendix A. Letting $^\intercal$ denotes the transpose operator, we have also defined a column vector $\boldsymbol{\varphi}^b_{\boldsymbol{\vartheta}} = [\varphi(\mathbf{x}_1;\boldsymbol{\vartheta}), \ldots, \varphi(\mathbf{x}_{N_b};\boldsymbol{\vartheta})]^\intercal \in \mathbf{R}^{N_b}$, whose components are also elements of the column vector $\boldsymbol{\varphi}_{\boldsymbol{\vartheta}} = [\varphi(\mathbf{x}_1;\boldsymbol{\vartheta}), \ldots, \varphi(\mathbf{x}_N;\boldsymbol{\vartheta})]^\intercal \in \mathbf{R}^N$; the column vector $\mathbf{f} = [f(\mathbf{x}_1), \ldots, f(\mathbf{x}_{N-N_b})]^\intercal \in \mathbf{R}^{N-N_b}$ with function values on the interior points; and the column vector $\mathbf{g} = [g(\mathbf{x}_1), \ldots, g(\mathbf{x}_{N_b})]^\intercal \in \mathbf{R}^{N_b}$ with function values on the boundary points. In the case of other kinds of boundary conditions, a corresponding boundary operator can be applied to $\boldsymbol{\varphi}_{\boldsymbol{\vartheta}}$ to enforce the boundary condition.

4 Theoretical Foundation of the Proposed Algorithm

The classical machine learning theory concerns with characterizations of the approximation error, optimization error estimation, and generalization error analysis. For the proposed PDE solver, the approximation theory involves characterizing: 1) the error of the DM-based discrete approximation of the differential operator on manifolds, and 2) the error of NNs for approximating the PDE solution. In the optimization algorithm, a numerical minimizer (denoted as $\boldsymbol{\vartheta}_N$) provided by a certain algorithm might not be a global minimizer of the empirical risk minimization in (8) and (9). Therefore, designing an efficient optimization algorithm such that the optimization error $|J_{S,\epsilon}(\boldsymbol{\vartheta}_N) - J_{S,\epsilon}(\boldsymbol{\vartheta}_S)| \approx 0$ is important. In the generalization analysis, the goal is to quantify the error defined as $\|u - \varphi(\bullet, \boldsymbol{\vartheta}_S)\|^2_{L^2(\pi)}$ over the distribution $\mathbf{x} \sim \pi$ that is unknown.

In approximation theory, the error analysis of DM is relatively well-developed, while the error of NNs is still under active development. Recently, there are two directions have been proposed to characterize the approximation capacity of NNs. The first one characterizes the approximation error in terms of the total number of parameters in an NN [97, 98, 72, 30, 74, 99, 77, 39, 89]. The second one quantifies the approximation error in terms of NN width and depth [84, 62, 85, 87, 86, 96]. In real applications, the width and depth of NNs are the required hyper-parameters to decide for numerical implementation instead of the total number of parameters. Hence, we will develop the approximation theory for the proposed PDE solver adopting the second direction.

In optimization theory, for regression problems without regularization (e.g., the right equation of (7)), it has been shown in [50, 22, 69, 23, 63] that GD and SGD algorithms can converge to a global minimizer of the empirical loss function under the assumption of over-parametrization (i.e., the number of parameters are much larger than the number of samples). However, existing results for regression problems cannot be applied to the minimization problem corresponding to PDE solvers, which is much more difficult due to differential operators and boundary operators. A preliminary attempt was conducted in [65], but the results in [65] cannot be directly applied to our minimization problem in (8) and (9). Below, we will develop a new analysis to show that GD can identify a global minimizer of (8).

The generalization analysis aims at quantifying the convergence of the generalization error $\|u - \varphi(\bullet;\boldsymbol{\vartheta}_S)\|_{L^2(\pi)}$. Let $u(X) \in \mathbf{R}^N$ be a column vector representing the evaluation of u on the training data set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ and this notation is used similarly for other functions. Beyond the identification of $\boldsymbol{\vartheta}_S$ (a global minimizer of the empirical loss), which is addressed in the optimization theory, a typical approach is to estimate the difference between $\|u - \varphi(\bullet;\boldsymbol{\vartheta}_S)\|_{L^2(\pi)}$ and $\|u(X) - \varphi(X;\boldsymbol{\vartheta}_S)\|_{L^2(\pi_N)}$ via statistical learning theories. There have been several papers for the generalization error analysis of PDE solvers. In [8, 42, 88, 64, 24, 47, 61], they show that minimizers of the empirical risk minimization can facilitate a small generalization error if these minimizers satisfy a certain norm constraint (e.g., with a small parameter norm or corresponding to an NN with a small Lipschitz constant). However, it is still an open problem to design numerical optimization algorithms to identify such a minimizer. Another direction is to regularize the empirical risk minimization so that a global minimizer of the regularized loss can generalize well [65]. Nevertheless, there is no global convergence analysis of the optimization algorithm for the regularized loss, which suggests that there is no guarantee that one can practically obtain the global minimizer of the regularized loss that can generalize well.

The theoretical analysis in this paper focuses on the approximation and optimization perspectives of the proposed PDE solver to develop an error analysis of $\|u(X)-\varphi(X;\boldsymbol{\vartheta}_S)\|_{L^2(\pi_N)}$. In the discussion below, we restrict to the case of manifolds without boundaries to convey our main ideas for the theoretical analysis of the proposed algorithm. We further assume that the ReLU activation function, i.e., $\max\{x,0\}$, its power, e.g., ReLU^r, and FNNs are used in the analysis. An extension to other activation functions and network architectures might be possible. We will show that the numerical solution of the proposed solver is consistent with the ground truth in appropriate limits, assuming that a global minimizer of our optimization problem is obtainable. We will also show that GD can identify a global minimizer of our optimization problem for two-layer NNs when their width is sufficiently large. The optimization theory together with the parametrization error analysis forms the theoretical foundation of the convergence analysis of the proposed PDE solver on manifolds.

4.1 Parametrization Error

The proposed PDE solver on manifolds applies DM to parametrize differential operators on manifolds and uses an NN to parametrize the PDE solution. We will quantify the parametrization error due to these two ideas, assuming that a global minimizer of the empirical loss minimization in (8) is achievable via a certain numerical optimization algorithm, i.e., estimating $\|\mathbf{u}-\boldsymbol{\varphi}_S\|_{\ell^2(\pi_N)}$, where $\boldsymbol{\varphi}_S \in \mathbf{R}^N$ with the i-th entry as $\varphi(\mathbf{x}_i;\boldsymbol{\vartheta}_S)$ is the NN-solution of the PDE in (8).

Theorem 4.1 (Parametrization Error). *Let u be the solution of (1) with $0 < a_{\min} \le a(x) \le a_{\max}$ on $X = \{\mathbf{x}_1,\dots,\mathbf{x}_N\}$, randomly sampled from a distribution π on $M \subset [0,1]^n$, where M is a C^4-manifold with condition number τ_M^{-1}, volume V_M, and geodesic covering regularity G_M. For $u, \kappa \in C^4(M) \cap L^2(M)$ and $q \in C(M)$, where $q = d\pi/dV$, with probability higher than $1-N^{-2}$,*

$$\|u(X)-\varphi(X;\boldsymbol{\vartheta}_S)\|_{L^2(\pi_N)} = O\left(\epsilon + \left(\frac{\log(N)}{N}\right)^{\frac{1}{2}}\epsilon^{-2-\frac{d}{4}} + \left(\frac{\log(N)}{N}\right)^{\frac{1}{2}}\epsilon^{2-\frac{d}{4}} + N^{1/2}\epsilon^{-1}m^{-\frac{8}{d\ln(n)}}L^{-\frac{8}{d\ln(n)}}\right), \tag{10}$$

as $\epsilon \to 0$, after $N \to \infty$ and m or $L \to \infty$. Hence, $\lim_{\epsilon\to 0}\lim_{N\to\infty}\lim_{m\to\infty}\|u(X)-\varphi(X;\boldsymbol{\vartheta}_S)\|_{L^2(\pi_N)} = 0$. Here $\varphi(\mathbf{x};\boldsymbol{\vartheta}_S)$ has width $O(n\ln(n)m\log(m))$ and depth $O(L\log(L)+\ln(n))$ with $m \in \mathbf{N}^+$ and $L \in \mathbf{N}^+$ as two integer hyper-parameters.

In (10) and throughout this paper, the big-oh notation means $O(f,g) := O(f)+O(g)$, as $f,g \to 0$. The first three error terms of (10) come from the DM discretization, whereas the last error term is due to the approximation property of NNs. The sequence of convergence in the big-O notation is due to the fact that $\epsilon > 0$ is fixed when we analyze the discretization error. For example, the first error rate in (10) corresponds to the error induced by the integral operator approximation in (3), whereas the third error rate in (10) corresponds to the discretization of L_ϵ using $\mathbf{L}_\epsilon$ for which one deduces error rate as $N \to \infty$ with fixed ϵ. The same reasoning applies to the NN approximation.

In (10), in the case of uniformly sampled data, the second error term, induced by the estimation of the sampling density q, becomes irrelevant and the factor $N^{1/2}$ in the last error term disappears due to symmetry. In such a case, the number of data points needed to achieve order-ϵ of the DM discretization is $N \ge N_0 = O(\epsilon^{-\frac{6+d}{2}})$, obtained by balancing the first and third error terms in (10). The NN width to achieve the same accuracy is $m \ge m_0 = O(\epsilon^{-\frac{d\ln(n)}{4}})$, obtained by balancing the first and last error terms in (10).

Before we prove Theorem 4.1, let us review relevant results that will be used for the proof.

The Parametrization Error of DM. For reader's convenience, we briefly summarize the pointwise error bound of the discrete estimator, which has been reported extensively (see e.g., [16, 90, 9, 25]). Our particular interest is to quantify the error induced by the matrix $\mathbf{L}_{a,\epsilon} := \mathbf{a} + \mathbf{L}_\epsilon$, where $\mathbf{L}_\epsilon$ is defined right after (3) and $\mathbf{a}$ is a diagonal matrix with diagonal entries $\mathbf{a}_{ii} = a(\mathbf{x}_i)$.

First, let us quantify the Monte-Carlo error of the discretization of the integral operator, i.e., the error for introducing L_ϵ in (3). In particular, using the Chernoff bound (see Appendix B.2 in [9] or Appendix A in [36]), for any $\mathbf{x}_i \in X$ and any fixed $\epsilon, \eta > 0$, and $u \in L^2(M)$, we have

$$\mathbf{P}(|(\mathbf{L}_\epsilon \mathbf{u})_i - L_\epsilon u(\mathbf{x}_i)| > \eta) < 2\exp\left(-C\frac{\eta^2\epsilon^{d/2+1}N}{\|\nabla_g u(\mathbf{x}_i)\|^2 q(\mathbf{x}_i)+1}\right),$$

for some constant C that is independent of ϵ and N. This version of Chernoff bound accounts for the variance error and yields similar rate as in [46], which uses the Bernstein inequality. Choosing $N^2 = \frac{1}{2}\exp\left(C\frac{\eta^2\epsilon^{d/2+1}N}{\|\nabla_g u(\mathbf{x}_i)\|^2 q(\mathbf{x}_i)+1}\right)$, one can deduce that

$$\eta = C^{-1/2}\left(\frac{\log(2N^2)}{N}\right)^{1/2}\epsilon^{-1/2-d/4}\|\nabla_g u(\mathbf{x}_i)\|q(\mathbf{x}_i)^{-1/2},$$

which means that with probability greater than $1-N^{-2}$,

$$(\mathbf{L}_\epsilon \mathbf{u})_i = L_\epsilon u(\mathbf{x}_i) + O\left(\frac{\|\nabla_g u(\mathbf{x}_i)\| q(\mathbf{x}_i)^{-1/2} (\log(N))^{1/2}}{N^{1/2}\epsilon^{1/2+d/4}}\right),$$

as $N \to \infty$. When the density $q = d\pi/dV$ is non-uniform, one can use the same argument (e.g., see Appendix B.1 in [9]) to deduce the error induced by the estimation of the density, which is of order $O(q(\mathbf{x}_j)^{1/2}(\log(N)/N)^{1/2}\epsilon^{-2-d/4})$ with probability $1-N^{-2}$, to ensure a density estimation of order-ϵ^2. Together with (3), we have the following pointwise error estimate.

Lemma 4.1. *Let $u, \kappa \in C^3(M) \cap L^2(M)$ and $q \in C(M)$, where $M \subseteq \mathbf{R}^n$ is a d-dimensional closed C^3-submanifold, then for any $\mathbf{x}_i \in X$, with probability higher than $1-N^{-2}$,*

$$(\mathbf{L}_{a,\epsilon}\mathbf{u})_i - L_a u(\mathbf{x}_i) = (\mathbf{L}_\epsilon \mathbf{u})_i - L_a u(\mathbf{x}_i) = O\left(\epsilon, \frac{q(\mathbf{x}_i)^{1/2}(\log(N))^{1/2}}{N^{1/2}\epsilon^{2+d/4}}, \frac{\|\nabla_g u(\mathbf{x}_i)\| q(\mathbf{x}_i)^{-1/2}}{N^{1/2}\epsilon^{1/2+d/4}}(\log(N))^{1/2}\right), \tag{11}$$

as $\epsilon \to 0$ after $N \to \infty$.

If we allow for $u, \kappa \in C^4(M)$, then by Lemma 3.3 in [45], we can replace the first-order error term in (11) by $R_u(x)\epsilon^{4\beta-1}$, where $0 < \beta < 1/2$ such that

$$\|R_u\|_{L^2(M)} \le C\|u\|_{H^4(M)}\|\pi\kappa\|^2_{C^4(M)} < \infty$$

for some constant $C > 0$ for $u, \kappa \in C^4(M)$. The term R_u will also appear in the second-error term as well since this error bound is to ensure the density estimation to achieve $O(\epsilon^2)$. Also, with the given assumptions, it is immediate to show that $\|R_u q^{1/2}\|^2_{L^2(\pi)}, \|R_u\|^2_{L^2(\pi)} < \infty$.

For a fixed $\epsilon > 0$ and using the fact that $\lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^N f(\mathbf{x}_i)^2 = \int_M f(x)^2 q(x) dV(x) = \|f\|^2_{L^2(\pi)}$, we have,

$$\begin{aligned}
\lim_{N\to\infty} \|\mathbf{L}_{a,\epsilon}\mathbf{u} - L_a u(X)\|^2_{L^2(\pi_N)} &= \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^N |(\mathbf{L}_{a,\epsilon}\mathbf{u})_i - L_a u(x_i)|^2 \\
&\le C_1\epsilon^{8\beta-2} \lim_{N\to\infty} \frac{1}{N}\sum_{i=1}^N R_u(x_i)^2 + C_2\epsilon^{-4-2d} \lim_{N\to\infty} \frac{1}{N^2}\sum_{i=1}^N (R_u(x_i)^2 q(x_i)) \\
&\quad + C_3\epsilon^{-1-2d} \lim_{N\to\infty} \frac{1}{N^2}\sum_{i=1}^N \|\nabla_g u(x_i)\|^2 (q(x_i)^{-1}) \\
&= C_1\epsilon^{8\beta-2}\|R_u\|^2_{L^2(\pi)} + \lim_{N\to\infty} \frac{1}{N}\left(C_2\epsilon^{-4-2d}\|R_u q^{1/2}\|^2_{L^2(\pi)} + C_3\epsilon^{-1-2d}\|u\|^2_{H^1(M)}\right) \\
&= C_1\epsilon^{8\beta-2}\|R_u\|^2_{L^2(\pi)}.
\end{aligned}$$

To conclude, we have the following lemma.

Lemma 4.2. *Let $u, \kappa \in C^4(M)$ and $q \in C(M)$ and let $M \in \mathbf{R}^n$ be a d-dimensional, closed, C^3-submanifold. Then,*

$$\lim_{\epsilon\to 0}\lim_{N\to\infty} \|\mathbf{L}_{a,\epsilon}\mathbf{u} - L_a u(X)\|_{L^2(\pi_N)} = 0, \tag{12}$$

almost surely.

This consistency estimate only holds when the limits are taken in the sequence as above.

The Parametrization Error of NNs. Let us denote the best possible empirical loss as $J_{S,\epsilon}(\vartheta_S)$, which depends only on the NN model and is independent of the optimization algorithm to solve the empirical loss minimization in (8), since ϑ_S is a global minimizer of the empirical loss. The estimation of $J_{S,\epsilon}(\vartheta_S)$ can be derived from deep network approximation theory in Theorem 1.1 of [62] and its corollary in [21]. The (nearly optimal) error bound in Theorem 1.1 of [62] focuses on approximating functions in $C^s([0,1]^n)$ and hence suffers from the curse of dimensionality; namely, the total number of parameters in the ReLU FNN scales exponentially in n to achieve the same approximation accuracy. Fortunately, our PDE solution is only defined on a d-dimensional manifold embedded in $[0,1]^n$ with $d \ll n$. By taking advantage of the low-dimensional manifold, a corollary of Theorem 1.1 in [62] was proposed in [21] following the idea in [84] to conquer the curse of dimensionality. For completeness, we quote this corollary below.

Lemma 4.3 (Proposition 4.2 of [21]). *Given $m, L \in \mathbf{N}^+$, $\mu \in (0,1)$, $v \in (0,1)$. Let $M \subset \mathbf{R}^n$ be a compact d-dimensional Riemannian submanifold having condition number τ_M^{-1}, volume V_M, and geodesic covering regularity G_M, and define the μ-neighborhood as $M_\mu := \{\mathbf{x} \in \mathbf{R}^n : \inf_{\mathbf{y}\in M} \|\mathbf{x}-\mathbf{y}\|_2 \le \mu\}$. If $u \in C^s(M_\mu)$ with $s \in \mathbf{N}^+$, then there exists a ReLU FNN φ with width $17 s^{d_v+1} 3^{d_v} d_v (m+2)\log_2(8m)$ and depth $18 s^2 (L+2)\log_2(4L) + 2d_v$ such that for any $\mathbf{x} \in M_\mu$,*

$$|\varphi(\mathbf{x}) - u(\mathbf{x})| \le 8\|u\|_{C^s(M_\mu)} \mu^{\left(1-\sqrt{d_v}+1\right)} (1-v)^{-1} n^{\sqrt{d_v}} + 170(s+1)^{d_v} 8^s (1-v)^{-1} \|u\|_{C^s(M_\mu)} m^{-2s/d_v} L^{-2s/d_v}, \quad (13)$$

where $d_v := \lfloor d \ln n V_M G_M \tau_M^{-1} / v^2 \rfloor / v^2 \rfloor = \lfloor d \ln(n/v)/v^2 \rfloor$ is an integer with $d < d_v < n$.

In Lemma 4.3, ReLU FNNs are used while in practice the learnable linear combination of a few activation functions might boost the numerical performance of neural networks [59]. The extension of Lemma 4.3 to the case of multiple kinds of activation functions is interesting future work. Lemma 4.3 can provide a baseline characterization to the approximation capacity of FNNs when ReLU is one of the choices of activation functions for a learnable linear combination. Hence, the following error estimation is still true for NNs constructed with the learnable linear combination of a few activation functions.

Using the same argument as the extension lemma for smooth functions (see e.g., Lemma 2.26 in [54]), one can extend any $u \in C^4(M)$ to $u \in C^4(M_{\mu_0})$, for any C^4 manifold $M \subseteq \mathbf{R}^n$ and a positive μ_0. Therefore, by Lemma 4.3, there exists a ReLU FNN φ with width $O(n\ln(n) m\log(m))$ and depth $O(L\log(L) + \ln(n))$ such that

$$|u(\mathbf{x}) - \varphi(\mathbf{x})| \le C_{M,d,v} \|u\|_{C^3(M_{\mu_0})} \left[\mu^2 + \frac{n}{\ln n} + n m^{-8/(d\ln(n))} L^{-8/(d\ln(n))} \right] \quad \text{for all } \mathbf{x} \in M,$$

for any $\mu \in (0, \mu_0)$, where $C_{M,d,v}$ is a constant depending only on M, d, and v. Note that the curse of dimensionality has been lessened in the above approximation rate. Therefore, by taking $\mu \to 0$, we have the following error estimation

$$\|\mathbf{u} - \boldsymbol{\varphi}\|_{L^2(\pi_N)} = O(m^{-8/(d\ln(n))} L^{-8/(d\ln(n))}), \quad (14)$$

where the prefactor depends on M, d, v, $\|u\|_{C^3(M_{\mu_0})}$, and n. By (11) and (14),

$$\begin{aligned} \|\mathbf{L}_{a,\epsilon}\boldsymbol{\varphi}_S - \mathbf{f}\|_{L^2(\pi_N)} &\le \|\mathbf{L}_{a,\epsilon}\boldsymbol{\varphi} - \mathbf{f}\|_{L^2(\pi_N)} \le \|\mathbf{L}_{a,\epsilon}\boldsymbol{\varphi} - \mathbf{L}_{a,\epsilon}\mathbf{u}\|_{L^2(\pi_N)} + \|\mathbf{L}_{a,\epsilon}\mathbf{u} - \mathrm{L}_a u(X)\|_{L^2(\pi_N)} \\ &\le \|\mathbf{L}_{a,\epsilon}\|_2 \|\boldsymbol{\varphi} - \mathbf{u}\|_{L^2(\pi_N)} + O\left(\epsilon^{-2-d/2}\left[\frac{\log(N)}{N}\right]^{1/2}\right) + O\left(\epsilon^{-2-d/4}\left[\frac{\log(N)}{N}\right]^{1/2}\right) \\ &= \|\mathbf{L}_{a,\epsilon}\|_2 O(m^{-8/(d\ln(n))} L^{-8/(d\ln(n))}) + O\left(\epsilon^{-2-d/2}\left[\frac{\log(N)}{N}\right]^{1/2}\right) + O\left(\epsilon^{-2-d/4}\left[\frac{\log(N)}{N}\right]^{1/2}\right), \end{aligned} \quad (15)$$

where, for simplicity, we suppressed the functional dependence on $\mathbf{x}$ in the second error term above.

Recall that $\mathbf{L}_{a,\epsilon} = \|\mathbf{a}\| + \mathbf{L}_\epsilon$, where $\mathbf{a}$ is a diagonal matrix with diagonal entries $0 < a_{\min} \le a(x_i) \le a_{\max}$ and $\mathbf{L}_\epsilon \mathbf{1} = \mathbf{0}$. By definition, $\mathbf{L}_\epsilon$ is diagonally dominant with diagonal negative entries and non-diagonal positive entries. This means

$$\|\mathbf{L}_{a,\epsilon}\|_\infty = \max_{1\le j\le N} \left[a(\mathbf{x}_j) - \mathbf{L}_{\epsilon,jj} + \sum_{i=j} \mathbf{L}_{\epsilon,ij} \right] = \max_{1\le j\le N} \left[a(\mathbf{x}_j) - 2\mathbf{L}_{\epsilon,jj} \right] = a_{\max} + C\epsilon^{-1},$$

for some constant C that depends on $\|\kappa\|_\infty$. Therefore, $\|\mathbf{L}_{a,\epsilon}\|_2 \le N^{1/2}\|\mathbf{L}_{a,\epsilon}\|_\infty \le C N^{1/2}\epsilon^{-1}$. Plugging this to (15), we obtain

$$\|\mathbf{L}_{a,\epsilon}\boldsymbol{\varphi}_S - \mathbf{f}\|_{L^2(\pi_N)} = O\left(\epsilon^{-2-d/2}\left[\frac{\log(N)}{N}\right]^{1/2}, \epsilon^{-2-d/4}\left[\frac{\log(N)}{N}\right]^{1/2}, N^{1/2}\epsilon^{-1} m^{-8/(d\ln(n))} L^{-8/(d\ln(n))}\right), \quad (16)$$

as $\epsilon \to 0$ after $N \to \infty$, and m or $L \to \infty$. This concludes the upper bound for the best possible empirical loss.

Proof of Theorem 4.1. Now we derive the overall parametrization error considering DM and NN parametrization together to prove Theorem 4.1. Since $\mathbf{L}_\epsilon$ is an unnormalized discrete Graph-Laplacian matrix that is semi-negative definite, for $a \ge a_{\min} > 0$, one can see that,

$$\langle \boldsymbol{\xi}, \mathbf{L}_{a,\epsilon}\boldsymbol{\xi}\rangle_{L^2(\pi_N)} = \langle \boldsymbol{\xi}, \mathbf{L}_\epsilon \boldsymbol{\xi}\rangle_{L^2(\pi_N)} + \langle \boldsymbol{\xi}, \mathbf{a}\boldsymbol{\xi}\rangle_{L^2(\pi_N)} \ge \|a_{\min}\|\boldsymbol{\xi}\|^2_{L^2(\pi_N)},$$

for any $\boldsymbol{\xi} \in L^2(\pi_N)$. Letting $\boldsymbol{\xi} = \mathbf{u} - \boldsymbol{\varphi}_S$, we have,

$$a_{\min}\|\mathbf{u} - \boldsymbol{\varphi}_S\|^2_{L^2(\pi_N)} \le \langle \mathbf{u} - \boldsymbol{\varphi}_S, \mathbf{L}_{a,\epsilon}(\mathbf{u} - \boldsymbol{\varphi}_S)\rangle_{L^2(\pi_N)} \le \|\mathbf{L}_{a,\epsilon}(\mathbf{u} - \boldsymbol{\varphi}_S)\|_{L^2(\pi_N)} \|\mathbf{u} - \boldsymbol{\varphi}_S\|_{L^2(\pi_N)}$$

and with probability higher than $1-N^{-2}$,

$$\begin{aligned}
\|\mathbf{u}-\boldsymbol{\varphi}_S\|_{L^2(\pi_N)} &\le \frac{1}{a_{\min}}\|\mathbf{L}_{a,\epsilon}(\mathbf{u}-\boldsymbol{\varphi}_S)\|_{L^2(\pi_N)} \\
&\le \frac{1}{a_{\min}}\left(\|\mathbf{L}_{a,\epsilon}\mathbf{u}-\mathbf{f}\|_{L^2(\pi_N)}+\|\mathbf{L}_{a,\epsilon}\boldsymbol{\varphi}_S-\mathbf{f}\|_{L^2(\pi_N)}\right), \\
&\le \frac{1}{a_{\min}}\left(\|\mathbf{L}_{a,\epsilon}\mathbf{u}-\mathbf{L}_a u(X)\|_{L^2(\pi_N)}+\|\mathbf{L}_{a,\epsilon}\boldsymbol{\varphi}_S-\mathbf{f}\|_{L^2(\pi_N)}\right), \\
&= O\left(\epsilon,\ \left(\frac{\log(N)}{N}\right)^{\frac{1}{2}}\epsilon^{-2-\frac{d}{4}},\ \left(\frac{\log(N)}{N}\right)^{\frac{1}{2}}\epsilon^{-2-\frac{1}{4}},\ N^{1/2}\epsilon^{-1}m^{-8/(d\ln(n))}L^{-8/(d\ln(n))}\right),
\end{aligned} \tag{17}$$

as $\epsilon\to 0$ after $N\to\infty$, and m or $L\to\infty$. Here, we have used (11) and (16) in the last equality above. Since the inequality in (13) is valid uniformly, together with (12), we achieve the convergence of the limit and the proof is completed. □

We should remark that if $\mathbf{L}_{a,\epsilon}\boldsymbol{\varphi}_S=\mathbf{f}$ is satisfied pointwise (e.g., solution induced by the classical finite-difference approximation), then the residual error in (16) is zero and the proof is nothing but the classical Lax-equivalence argument of consistency in (11) and stability implies convergence of the solution.

4.2 Optimization Error

In the parametrization error analysis, we have assumed that a global minimizer of the empirical loss minimization in (8) is achievable. In practice, a numerical optimization algorithm is used to solve this optimization problem and the numerical minimizer might not be equal to a global minimizer. Therefore, it is important to investigate the numerical convergence of an optimization algorithm to a global minimizer. The neural tangent kernel analysis originated in [50] and further developed in [4, 3] has been proposed to analyze the global convergence of GD and SGD for the least-squares empirical loss function in (8). In the situation of solving PDEs, the global convergence analysis of optimization algorithms is vastly open. In [65], it was shown that GD can identify a global minimizer of the least-squares optimization for solving second-order linear PDEs with two-layer NNs under the assumption of over-parametrization. In this paper, we will extend this result in the context where the differential operator in the loss function is replaced by a discrete and approximate operator, $\mathbf{L}_{a,\epsilon}$, applied to the NN solution (e.g., see (8) and (9)). Furthermore, the activation function considered here is ReLU^r, which is more general than the activation function in [65]. The extension to deeper NNs follows the analysis in [3] and is left as future work.

To establish a general theorem for various applications, we consider the following emprical risk $R_S(\boldsymbol{\vartheta})$ with an over-parametrized two-layer NN optimized by GD:

$$R_S(\boldsymbol{\vartheta}) := \frac{1}{2N}(\boldsymbol{A}\varphi(X;\boldsymbol{\vartheta})-f(X))^{\top}(\boldsymbol{A}\varphi(X;\boldsymbol{\vartheta})-f(X)), \tag{18}$$

where $X:=\{\boldsymbol{x}_i\}_{i=1}^N$ is a given set of samples in $[0,1]^n$ of an arbitrary distribution (π in our specific application); $\boldsymbol{A}$ is a given matrix of size $N\times N$; and the two-layer NN used here is constructed as $\varphi(\boldsymbol{x};\boldsymbol{\vartheta})=\sum_{k=1}^m a_k\sigma(\boldsymbol{w}_k^{\top}\boldsymbol{x})$, where for $k\in[m]:=\{1,\dots,m\}$, $a_k\in\mathbf{R}$, $\boldsymbol{w}_k\in\mathbf{R}^n$, $\boldsymbol{\vartheta}=\mathrm{vec}\{a_k,\boldsymbol{w}_k\}_{k=1}^m$, and $\sigma(x)=\mathrm{ReLU}^r(x)$, i.e., the r-th power of the ReLU activation function. Note that the matrix $\boldsymbol{A}$ in our specific application is $\mathbf{L}_{a,\epsilon}$ for (8); $\boldsymbol{A}$ is a block-diagonal matrix for (9) with one block for the differential equation and another block for the boundary condition; $\boldsymbol{A}$ is also a block-diagonal matrix when a regularization term $\|\boldsymbol{\varphi}_{\boldsymbol{\vartheta}}\|^2_{L^2(\pi_N)}$ is applied to either (8) or (9). In the remainder of this section, we use the notation $\boldsymbol{A}$ since the result holds in general under some assumption discussed below. Without loss of generality, throughout our analysis, we also assume $|f|\le 1$, since the PDE defined on a compact domain can be normalized.

Adopting the neuron tangent kernel point of view [50], in the case of a two-layer NN with an infinite width, the corresponding kernel $k^{(a)}$ for parameters in the last linear transform is a function from $M\times M$ to $\mathbf{R}$ defined by $k^{(a)}(\boldsymbol{x},\boldsymbol{x}'):=\mathbf{E}_{\boldsymbol{w}\sim N(\mathbf{0},I)}g^{(a)}(\boldsymbol{w};\boldsymbol{x},\boldsymbol{x}')$, where $g^{(a)}(\boldsymbol{w};\boldsymbol{x},\boldsymbol{x}'):=[\sigma(\boldsymbol{w}^{\top}\boldsymbol{x})]^{\top}\sigma(\boldsymbol{w}^{\top}\boldsymbol{x}')$. This kernel evaluated at $N\times N$ pairs of samples leads to an $N\times N$ Gram matrix $\boldsymbol{K}^{(a)}$ with $K^{(a)}_{ij}=k^{(a)}(\boldsymbol{x}_i,\boldsymbol{x}_j)$. Our analysis requires the matrix $\boldsymbol{K}^{(a)}$ to be positive definite, which has been verified for regression problems under mild conditions on random training data $X=\{\boldsymbol{x}_i\}_{i=1}^N$ and can be generalized to our case. Hence, we assume this together with the non-singularity of $\boldsymbol{A}$ as follows for simplicity.

Assumption 4.1. *The smallest eigenvalue of $\boldsymbol{K}^{(a)}$, denoted as λ_S, and the smallest eigenvalue of $\boldsymbol{A}\boldsymbol{A}^{\top}$, denoted as $\lambda_{\boldsymbol{A}}$, are positive.*

Let κ_A denote the condition number of $A^\top A$. Our main result of the global convergence of GD for (18) is summarized in Theorem 4.2 below with a proof in Appendix B.

Theorem 4.2 (Global Convergence of GD: Two-Layer NNs)**.** *Let $\boldsymbol{\vartheta}(0) := \mathrm{vec}\{a_k(0), \boldsymbol{w}_k(0)\}_{k=1}^m$ at the GD initialization for solving* (18)*, where $a_k(0) \sim N(0, \gamma^2)$ and $\boldsymbol{w}_k(0) \sim N(\mathbf{0}, \boldsymbol{I}_n)$ with any $\gamma \in (0,1)$. Let λ_S and λ_A be positive constants in Assumption 4.1. For any $\delta \in (0,1)$, if $m = \Omega(\kappa_A \mathrm{poly}(N, r, n, \frac{1}{\delta}, \frac{1}{\lambda_S}))$, then with probability at least $1-\delta$ over the random initialization $\boldsymbol{\vartheta}(0)$, we have, for all $t \geq 0$, $R_S(\boldsymbol{\vartheta}(t)) \leq \exp\left(-\frac{m\lambda_S\lambda_A t}{N}\right) R_S(\boldsymbol{\vartheta}(0))$.*

For the estimate of $R_S(\boldsymbol{\vartheta}(0))$, see Lemma B.4. In particular, if $\gamma = O(\frac{1}{m(\log m)^2})$, then $R_S(\boldsymbol{\vartheta}(0)) = O(1)$. For two-layer NNs, Theorem 4.2 shows that, as long as the NN width $m = \Omega(\kappa_A \mathrm{poly}(N, r, n, \frac{1}{\delta}, \frac{1}{\lambda_S}))$, GD can identify a global minimizer of the empirical risk minimization in (18). For a quantitative description of $O(\kappa_A \mathrm{poly}(N, r, n, \frac{1}{\delta}, \frac{1}{\lambda_S}))$, see (42) in Appendix B. In the case of FNNs with L layers, following the proof in [3], one can show the global convergence of GD when $m = \Omega(\kappa_A \mathrm{poly}(L, N, r, n, \frac{1}{\delta}, \frac{1}{\lambda_S}))$, which is left as future work.

5 Numerical Examples

We numerically demonstrate the effectiveness and practicability of our proposed NN-based PDE solver on six examples, ranging from simple manifolds to unknown rough surfaces. In the numerical experiments on simple manifolds, we will compare the accuracy of NN and DM (or VBDM) solutions not only on training data but also on new data points. Since DM (or VBDM) only approximates the PDE solution evaluated on the training data, we will consider the classical Nyström extension [75] as a means to interpolate the approximate PDE solutions on new data points. We will show that the NN solution provides robustly more accurate generalization on new data points with lower complexity once the training is completed. To demonstrate the advantage of the manifold assumption, we demonstrate the ability of the NN-based solver to deal with PDE on manifolds of high co-dimension by validating it on a three-dimensional manifold embedded in $\mathbf{R}^{12}$. We will see that the NN-based solver can obtain a more accurate solution than DM. Finally, we will verify the performance on unknown (and rough) surfaces. In these numerical experiments, unlike the existing works reported in [83, 58, 78], we do not smooth the surfaces. To avoid singularities induced by the original data set in these rough surfaces, we resample the data points using the Marching Cubes algorithm [60] that is available through Meshlab [71]. Since the analytical solutions are unknown in this configuration, we will compare the NN solutions with the finite element method (FEM) solutions obtained from the FELICITY FEM MATLAB toolbox [93].

The remainder of this section is organized as follows. In Section 5.1, we give an overview of the implementation detail of the neural-network regression. We will supplement this section with a list of detailed parameters used in each numerical example in Appendix C. In Section 5.2, we compare the NN generalization with the Nyström based interpolation method on simple manifolds. In Section 5.3, we examine the accuracy of the approach on a manifold with high codimension. In Section 5.4, we benchmark the numerical performances against the FEM solutions on unknown (and rough) surfaces.

5.1 Experimental design

In each of the numerical examples below, we find NN regression solution to the elliptic PDE in (1) embedded in $\mathbf{R}^n$. Using the notation from previous section, given point cloud data $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N \in \mathbf{R}^n\}$, we solve the PDE by optimizing the least-square problem given by,

$$\arg\min_{\boldsymbol{\vartheta}} \frac{1}{2}\|(\mathbf{a} + \mathbf{L}_\epsilon)\boldsymbol{\varphi}_{\boldsymbol{\vartheta}} - \mathbf{f}\|^2_{L^2(\pi_N)} + \frac{\gamma}{2}\|\boldsymbol{\varphi}_{\boldsymbol{\vartheta}}\|^2_{L^2(\pi_N)} + \frac{\lambda}{2}\|\boldsymbol{\varphi}^b_{\boldsymbol{\vartheta}} - \mathbf{g}\|^2_{L^2(\pi_{N_b})}. \tag{19}$$

Here, we add a regularization term with $\gamma > 0$ that is needed to overcome the ill-posedness induced by $a = 0$ in the no boundary case (as discussed right after (8)). For manifolds with boundary, we also set $\lambda > 0$. The detailed choices of these parameters for each numerical experiments are reported in Appendix C.

Devices and environments. The experiments of DM are conducted on the workstation with 32× Intel(R) Xeon(R) CPU E5-2667 v4 @ 3.20GHz and 1 TB RAM and Matlab R2019a. The experiments of the NN-solver are conducted on the workstation with 16× Intel(R) Xeon(R) Gold 5122 CPU @ 3.60GHz and 93G RAM using Pytorch 1.0 and 1× Tesla V100.

Implementation detail for neural-network regression. We summarize notations and list the hyperparameter setting for each numerical example in Appendix C. In our implementation, we use a 3-hidden-layer FNN with the same width m per hidden layer and the smooth Polynomial-Sine activation function in [59]. Polynomial-Sine is

defined as $\alpha_1 \sin(\beta_1 x) + \alpha_2 x + \alpha_3 x^2$, where the parameters are trainable and initialized by normal distributions $\beta_1, \alpha_1 \sim N(1, 0.01), \alpha_2, \alpha_3 \sim N(0, 0.01)$ respectively. As we shall demonstrate, our proposed NN solver is not sensitive to the numerical choices (e.g., activation functions, network types, and optimization algorithms) but a more advanced algorithm design may improve the accuracy and convergence. Particularly in the high codimensional example (Section 5.3), we will compare the performance of ReLU FNNs and ReLU^3 FNNs trained by the gradient descent method and the performance of the Polynomial-Sine FNNs trained with the Adam optimizer [53]. Though the ReLU or ReLU^3 FNNs with the gradient descent method enjoy theoretical guarantees in our analysis, the Polynomial-Sine FNNs with Adam can generalize better, especially if the PDE solutions are in the class of sine or cosine functions. Numerically, we will employ the Polynomial-Sine activation and Adam optimizer in almost all of our examples and report it separately otherwise. In Adam, we use an initial learning rate of 0.01 for T iterations. The learning rate follows cosine decay with the increasing training iterations, i.e., the learning rate decays by multiplying a factor $0.5(\cos(\frac{\pi t}{T}) + 1)$, where t is the current iteration. We set the batch size to be the total number of training points unless otherwise stated. The NN results reported in this section are averaged over 5 independent experiments.

Time complexity of evaluating NN solution. Since some numerical experiments will compare the accuracy of the solutions (generalization) on new data points, we document the time complexity of the NN-based function evaluation on a new datum. First, we note that the total of addition and multiplication of a linear transformation from $\mathbf{R}^A$ to $\mathbf{R}^B$ in a hidden layer of NN is $(A + (A - 1) + 1) \times B = 2AB$, where $(A + (A - 1) + 1)$ represents the amount of computation required to compute a neuron, and the first A is the amount of multiplication, $(A - 1)$ is the amount of addition, and 1 is from adding a bias. As introduced in Section 3.1, we use a L-layer FNN with a uniform width m. The input and output sizes are n and 1, respectively. Then the complexity of NN is $2nm + 2(L - 1)m^2 + 2m$, excluding the cost of evaluating the activation function on each layer. In our numerical examples, we use $m = O(\pi N)$ to facilitate training and fix $L = 4$, so the time complexity of evaluating 1 datum becomes $O(N)$ when $n << N$.

Implementation detail for DM and VBDM. For efficient implementation, we use the k-nearest neighbor algorithm to avoid computing the distances of pair of points that are sufficiently large. The kernel bandwidth parameter, ϵ, is tuned empirically for the fixed-bandwidth DM. For well-sampled data (data distributed equal angle intrinsically) in our first four examples, for large enough k, we found accurate estimate with $\epsilon \sim N^{-2/d}$, which is much faster than the theoretical error bound $\epsilon \sim N^{-\frac{2}{6+d}}$ obtained by balancing the first and third error bounds in Theorem 4.1. For randomly sampled data, such as in our last two examples, we will use VBDM where ϵ is estimated by an auto-tuning method that was originally proposed in [17]. Among the fixed-bandwidth DM applications, we found that we can directly use the auto-tuned ϵ for the semi-torus example. While the auto-tuned method may not produce the optimal parameter accurate estimation for fixed-bandwidth DM, this scheme is quite robust for the variable bandwidth diffusion maps (VBDM) as documented in [9]. We should also point out that in VBDM, we also need to prescribe an additional parameter $k_2 < k$ to specify ρ_0 that is used to determine the variable bandwidth function ρ in (6). We document the choice of nearest neighbor parameter k, the parameter k_2, and bandwidth parameter ϵ resulting either from hand-tuning or auto-tuning in the Tables in Appendix C.

For a DM/VBDM solution, when the resulting problem is ill-posed (due to $\mathbf{a} = 0$ and no boundary), the resulting matrix $\mathbf{L}_\epsilon$ is singular. In such a case, we report the numerical result based on applying the pseudo inverse of $\mathbf{L}_\epsilon$ to a right-hand-side vector. If the rank of the matrix $\mathbf{L}_\epsilon$ is r, then $O(N r^2)$ complexity is required to stably compute a truncated SVD of $\mathbf{L}_\epsilon$ [40], which can be used to form a truncated SVD of the pseudo inverse of $\mathbf{L}_\epsilon$. Then it takes $O(Nr)$ complexity to apply the pseudo inverse of $\mathbf{L}_\epsilon$ through its truncated SVD. The $O(N^2 r)$ complexity can be further reduced to $O(Nr^2)$ at the price of not stable to entrywise outliers in $\mathbf{L}_\epsilon$ [32, 95].

Nyström-based interpolation. We consider a Nyström-based interpolation [75] for comparing the accuracy of the estimates on new data points. For completeness of the discussion, we provide an algorithmic overview of the basic idea of Nyström interpolation to extend symmetric eigenvectors to new data point and discuss its implemetation for DM. For a more rigorous discussion that covers VBDM see e.g. Section 5.2 of [2].

Let $\mathbf{L} \in \mathbf{R}^{N \times N}$ be a symmetric positive definite matrix with eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_N > 0$ and we denote the associated eigenvectors by $\{\mathbf{v}_j\}_{j=1,\ldots,N}$. For convenience of discussion, we define a function $L : \mathbf{R}^n \times \mathbf{R}^n \to \mathbf{R}$ such that $L(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{L}_{ij}$. Effectively, the eigenvalue problem

$$\mathbf{L}\mathbf{v}_j = \lambda_j \mathbf{v}_j.$$

is a discrete approximation on $\mathbf{x} \in X$ to a continuous eigenvalue problem of the following integral operator,

$$\int_M L(\mathbf{x}, \mathbf{y}) v_j(\mathbf{y}) dV(\mathbf{y}) = \lambda_j v_j(\mathbf{x}),$$

assuming that the data set X is uniformly distributed for simplicity of discussion. Effectively, we approximate the function value $v_j(\mathbf{x}_i)$ with the ith component of the eigenvector $[\mathbf{v}_j]_i$. The main idea of Nyström extension is to

approximate the function value $v_j(\mathbf{x})$ on new data point $\mathbf{x}$ (not necessarily belong to the training data set X) using the equation above as follows,

$$v_j(\mathbf{x}) = \frac{1}{\lambda_j} \int_M L(\mathbf{x}, \mathbf{y}) v_j(\mathbf{y}) \, dV(\mathbf{y}).$$

Numerically, this can be achieved by the following approximation,

$$v_j(\mathbf{x}) \approx \frac{1}{\lambda_j} \sum_{i=1}^{N} L(\mathbf{x}, \mathbf{x}_i) [\mathbf{v}_j]_i . \tag{20}$$

For our problem, we are interested in applying this extension formula on an integral operator corresponding to the DM/VBDM asymptotic expansion that converges to the Laplace-Beltrami operator. In such a case, eigenfunctions of the Laplace-Beltrami operator can be approximated by eigenvectors of $\mathbf{L}^{ns} = \mathbf{D}^{-1}\mathbf{W}$, where $\mathbf{W}$ is defined as in (4) with $\kappa = 1$ and $Q = 1$ for uniform data, and $\mathbf{D}$ is a diagonal matrix with diagonal entries $\mathbf{D}_{ii} = \sum_{j=1}^{N} \mathbf{W}_{ij}$. For convenience of the discussion below, we define the kernel function W such that $\mathbf{W}_{ij} = W(\mathbf{x}_i, \mathbf{x}_j)$ as in (4).

We can now employ the Nyström extension to interpolate eigenfunctions of L associated with the symmetric discretization $\mathbf{L} = \mathbf{D}^{1/2}\mathbf{L}^{ns}\mathbf{D}^{-1/2} = \mathbf{D}^{-1/2}\mathbf{W}\mathbf{D}^{-1/2}$. Since $\boldsymbol{\varphi}_j := \mathbf{D}^{-1/2}\mathbf{v}_j$ is an eigenvector of $\mathbf{L}^{ns}$ corresponding to the same eigenvalue λ_j of $\mathbf{L}$, it is clear that given the function value $v_j(\mathbf{x})$ on a new datum $\mathbf{x}$, an extension attained by the formula in (20) with $L(\mathbf{x}, \mathbf{x}_i) := D(\mathbf{x})^{-1/2} W(\mathbf{x}, \mathbf{x}_i) D(\mathbf{x}_i)^{-1/2}$ where $D(\mathbf{x}) = \sum_{j=1}^{N} W(\mathbf{x}, \mathbf{x}_j)$, then the corresponding eigenfunction of the Laplace-Beltrami operator is approximated by $\varphi_j(\mathbf{x}) := D(\mathbf{x})^{-1/2} v_j(\mathbf{x})$. Similar idea applies to the Nyström extension corresponding to the VBDM asymptotic expansion with a slightly more complicated formula. For manifolds with homogeneous Dirichlet boundary conditions, we employ the extension using the truncated diffusion maps [76].

Given the ability to approximate the function values $\{\varphi_j(\mathbf{x})\}_{j=1}^{J}$ for any new datum $\mathbf{x}$, we evaluate the DM solution for the PDE problem by interpolating the projected solution to the leading J eigenfunctions of the Laplace-Beltrami operator as follows. Let $\mathbf{u} \in \mathbf{R}^N$ denotes the solution of the PDE attained by solving $(-\mathbf{a} + \mathbf{L}_\epsilon)\mathbf{u} = \mathbf{f}$ (either through direct or pseudo-inversion). Then we define the extension as,

$$u(\mathbf{x}) \approx \sum_{j=1}^{J} \mathbf{u}^{\top} \boldsymbol{\varphi}_j \varphi_j(\mathbf{x}), \tag{21}$$

where J is chosen sufficiently large such that the residual error is comparable to the interpolation error.

Time complexity of the Nyström extension. To employ the Nyström extension, we need to first attain eigenvectors $\{\boldsymbol{\varphi}_j\}_{j=1}^{J}$. Using the standard algorithm, one can attain the first J eigenvectors with time complexity of $O(Nk) + O(JkN)$, where k is the number of nearest neighbors; the first complexity term corresponds to the arithmetic in the construction of $\mathbf{L}$; and the second complexity term corresponds to the arithmetic in solving the first J eigenvalue problem. Subsequently, the arithmetic cost in attaining the coefficients in the expansion in (21) is $O(NJ)$ accounting for J inner products of vectors in $\mathbf{R}^N$, which is negligible compared to the computations in solving the eigenvectors. Once these computations are performed, we can now evaluate any data point with additional time complexity due to the arithmetic $O(NJ)$ per evaluation. This arithmetic cost includes the time complexity of $O(N)$ in the extension formula in (20), not accounting for the cost of evaluating the kernel function W, and the use of J eigenfunctions in (21) is of $O(JN)$.

Comparing the time complexities for evaluating the NN solution and Nyström interpolation, notice that the latter has an additional order-J, which has to be empirically determined in practice. As we pointed out above, the additional order-J appears in the estimation of leading J eigenvectors and the Nyström extension. In the remainder of this section, we shall see that in some examples, setting J to be of order 10 or 100 is sufficient to achieve the accuracy obtained by evaluating the NN solution. However, in some examples, we found that the accuracy of the NN solutions cannot be achieved by Nyström extension of DM solutions even with $J = 600$.

5.2 Comparison between Nyström extension and NN solution

In addition to the complexity comparisons reported in the previous section, we now compare the accuracy of the NN and the Nyström solutions. We will consider three instructive examples. In the first example, our goal is to validate the Nyström approach on a problem when there are no residuals of the projection in (21). In the second example, we want to elucidate the dependence of the Nyström approach on J when the PDE solution does not belong to the space span by the leading eigenfunctions of the Laplace-Beltrami operator. In this case, we will encounter

competing errors from the eigenvector estimation using small training data set, the choice of J, and interpolating error, in the overall generalization error. In the final example, we will elucidate the effect of boundary conditions on the Nyström-based interpolation method. In all of these experiments, we found that NN produces the most robust and accurate generalization.

5.2.1 2D Sphere

In this first example, we solve the elliptic PDE in (1) for $a = 0, \kappa = 1$ in a two-dimensional sphere embedded in $\mathbf{R}^3$ with an embedding function defined as,

$$\iota(t_1, t_2) = \begin{bmatrix} \cos t_1 \sin t_2 \\ \sin t_1 \sin t_2 \\ \cos t_2 \end{bmatrix} \in \mathbf{R}^3 \text{ for } \begin{matrix} 0 \le t_1 \le 2\pi \\ 0 \le t_2 \le \pi \end{matrix}. \tag{22}$$

For this simple geometry, one can verify that the Laplace-Beltrami operator has leading nontrivial eigenvalue of -2 with spherical harmonics eigenfunctions, $\varphi_1(\mathbf{x}) = x_1, \varphi_2(\mathbf{x}) = x_2, \varphi_3(\mathbf{x}) = x_3$ for $\mathbf{x} = (x_1, x_2, x_3)$. In this first test example, we choose the true solution of the PDE as $u(\mathbf{x}) = x_1 + x_2$ for $\mathbf{x} = (x_1, x_2, x_3)$, such that the expansion in (21) commits no residual error. Numerically, we will set $J = 20$ to ensure that even if the estimated eigenvectors corresponding to eigenvalue -2 are not exactly x_1, x_2, x_3 (up to an rotation), the projection still commits no residual error.

For this example, the matrix $\mathbf{L}_\epsilon$ is constructed by VBDM. We solve the PDE with the NN method by optimizing the least-square problem (19) with $\lambda = 0$ since this manifold has no boundary. The hyperparameter setting for the different N is reported in Table 6 in Appendix C. Table 1 displays the errors (in ℓ_∞ norm) as functions of N for the VBDM solution (which we refer to as the *training error*), and the testing errors of the Nyström-based solution projected on $J = 20$ eigenfunctions and the NN method. Here, the *testing errors* are computed against the true solution on 300^2 Gaussian Legendre quadrature nodes (which do not belong to the training data set). We can see that both Nystrom and NN methods achieve similar testing errors, that are also comparable to the VBDM training error.

	N	625	1225	2500	5041	10000
VBDM	training error	0.0352	0.0254	0.0178	0.0126	0.0090
Nystrom ($J = 20$)	testing error	0.0411	0.0268	0.0199	0.0128	0.0091
NN	testing error	0.0387	0.0271	0.0193	0.0128	0.0094

Table 1: (2D sphere) Error comparison of the PDE solutions among VBDM, Nystrom extension of the VBDM solution, and NN solution. The testing error is the ℓ_∞ error between the estimated solution and the true solution on 300^2 Gauss-Legendre quadrature points.

5.2.2 2D Torus

In this section, we consider solving the elliptic PDE in (1) for $a = 0$ on a two-dimensional torus embedded in $\mathbf{R}^3$ with an embedding function defined as,

$$\iota(t_1, t_2) = \begin{bmatrix} (2 + \cos t_1) \cos t_2 \\ (2 + \cos t_1) \sin t_2 \\ \sin t_1 \end{bmatrix} \in \mathbf{R}^3 \text{ for } \begin{matrix} 0 \le t_1 \le 2\pi \\ 0 \le t_2 \le 2\pi \end{matrix}. \tag{23}$$

Here t_1, t_2 denote the intrinsic coordinates. For this numerical experiment, we set the diffusion coefficient $\kappa(t_1, t_2) = 1.1 + \sin^2 t_1 \cos^2 t_2$, the true solution $u(t_1, t_2) = (\sin 2t_2 \cdot 2 \cos 2t_2 / (2 + \cos t_1)) \cos t_1$, and analytically compute the right hand side function f. The point cloud data $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ are well sampled (equal angles in intrinsic coordinates (t_1, t_2)). We solve the PDE by optimizing the least-square problem in (19) with $\lambda = 0$. The hyperparameter setting for the different N is reported in Table 7 in Appendix C.

Figure 1 shows the errors with the growth of training data N on the 2D full-torus. As in the previous example, the testing error is defined with ℓ_∞ error on 300^2 Gauss-Legendre quadrature points that are not in the training data set. One can see that the NN testing error is consistent with the DM training error, indicating a good generalization of new data points. We note that the accuracy of Nyström method heavily depends on the number of eigenfunctions used. When the number of eigenfunctions is small ($J = 20$), the errors on the testing points are large for all N. When the number of eigenfunctions is increased to $J = 100$, Nystrom has a good generalization when N is small ($N \le 1225$).

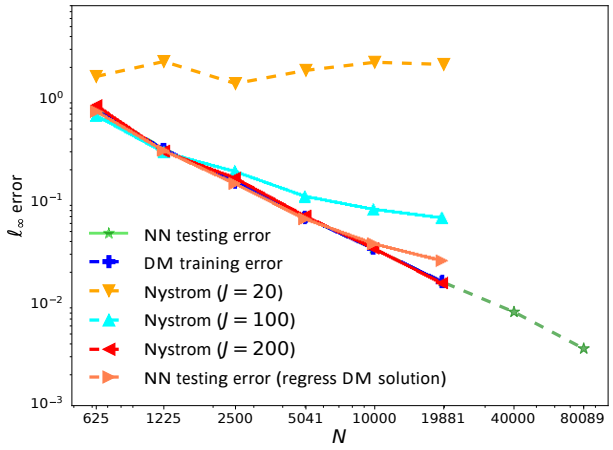


Figure 1: **2D-torus:** The comparison of the ℓ_∞ errors as functions of the number of training points N for DM and NN on a 2D torus embedded in $\mathbf{R}^3$. The results labeled as "(regress DM solution)" represent conducting NN regression directly on the DM solution.

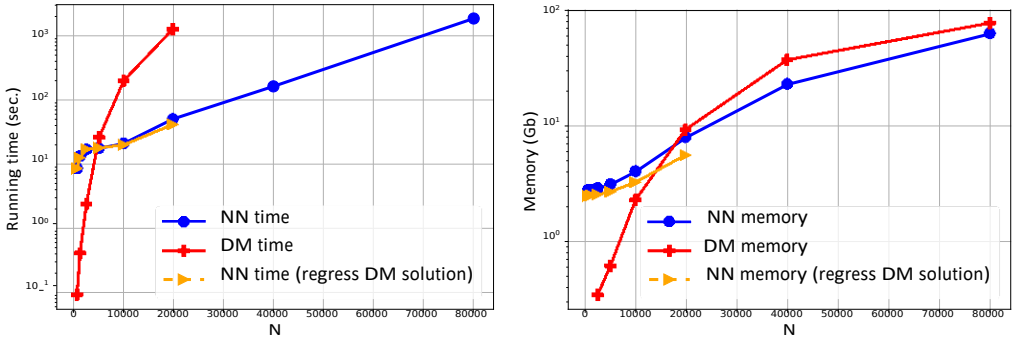
For $N \geq 2500$, the testing errors are smaller compared with $J = 20$ but they are still large compared with the DM training error. If we continue increasing J to 200, the testing errors of the large N ($N \geq 1225$) become consistent with the DM error. These results suggest that more eigenfunctions are able to suppress the residual error induced by the finite projection in (21). At the same time, when J is increased from 100 to 200, the error corresponds to $N = 625$ increases as well. This is not surprising since the estimation of eigenvectors corresponding to large eigenvalues are not so accurate when the number of data points is small. From these results, we conclude that while the Nyström method can be competitive with the NN solution, it depends crucially on the number of modes in (21), which ultimately also depends on the number of training data.

In Figure 1, we also show the testing error with NN solution for larger N, where the error continues to decrease with rate N^{-1}, which confirms with the choice of $\epsilon \sim N^{-1}$ for this well-sampled data (see Table 7). To obtain this solution with $N = 80089$, mini-batch training is used to minimize (19) since $\mathbf{L}_\epsilon \in \mathbf{R}^{80089\times 80089}$ is too large to compute in GPU directly. In our implementation, at each time, 8000 rows from $\mathbf{L}_\epsilon$ are randomly sampled and the submatrix $\mathbf{L}_{\text{sub}}$ of size 8000×80089 substitutes $\mathbf{L}_\epsilon$ in the loss (19). Then the network parameters $\boldsymbol{\vartheta}$ are updated for 100 iterations at each time. We repeat this procedure for 80 times. As for the DM, we do not pursue the results since the computational cost in solving the linear problem $\mathbf{L}_\epsilon \mathbf{u} = \mathbf{f}$ with singular $\mathbf{L}_\epsilon$ is very expensive as it involves pseudo-inversion algorithm.

To be more specific, we report the clock-time and memory cost for training DM and NN solutions in Figure 2. The clock-time for the pseudo-inverse of DM grows rapidly with the increasing N while that of NN remains much smaller when $N < 80089$. The rapid increase of NN clock-time for the case $N = 80089$ is attributed to the time consuming of the retrievals of the submatrix $\mathbf{L}_{\text{sub}}$ from $\mathbf{L}_\epsilon$. Memory in Figure 2 refers to the sum of GPU and RAM. When conducting pseudo-inverse in the DM, the Matlab occupies RAM to load the full matrix and do the computation. The NN-based solver utilizes RAM to load the sparse matrix and uses GPU memory for model training. From Figure 2, the NN solver has larger memory consumption than DM when N is small but the NN solver uses less memory than DM when N is large. Since we set the batch size to be the total number of training points for all cases except $N = 80089$, the memory consumption will significantly decrease if a mini-batch is used in Adam.

To obtain the NN solution to the manifold PDE, one could consider an *alternative method* consisting of two steps: First, use an inversion to solve the linear system, and then train a NN to regress the DM solution. While the alternative method may produce testing error similar to the DM method and the proposed NN method, we do not pursue it because the alternative method will cause a much higher cost even for small N due to its two-step approach. Figure 2 supplement the running time and memory usage of the NN training portion for the alternative method (labeled as "(regress DM solution)"). One can see that NN regression on the DM solutions does not significantly reduce running time or memory cost. Furthermore, Figure 1 provides testing error results for the alternative method. The testing error of the alternative method is slightly higher than the others, likely because we use the same training configuration for the alternative method. A better training configuration may maintain a testing error close to the DM training error. The NN training portion in the alternative method has comparable training time to the NN method but note that the alternative method includes two sequential steps. This means the total training time (DM solution time + NN training time to regress the DM solution) is much higher. Also, the memory requirement for the

alternative method is the maximum of the NN regression memory and DM memory, which is more costly in terms of memory.



(a) clock-time (b) memory

Figure 2: The comparison of clock-time, memory for DM and NN solvers on 2D torus embedded in $\mathbf{R}^3$. In the DM case, we also report the require memory, estimated by the system process-manager, to solve the problem for large N, which we did not pursue due to the excessive wall-clock time. The results labeled as "(regress DM solution)" represent conducting NN regression directly on the DM solution.

5.2.3 Semi-torus

We consider solving the PDE in (1) with $a = 0$ on a two-dimensional semi-torus M with a Dirichlet boundary condition. Here, the embedding function is the same as in (23) except that the range of t_2 is changed to $0 \le t_2 \le \pi$. While κ is chosen to be the same as before, we set $u(t_1, t_2) = \cos(t_1)\sin(2t_2)$ to be the solution with homogeneous Dirichlet boundary condition on $t_2 = 0, \pi$. We obtain the NN-based solution by solving the optimization problem in (19) with $\lambda = 5$ and no regularization $\gamma = 0$. The other hyperparameter setting can be found in Table 8 in Appendix C.

Figure 3 shows the errors as functions of training data N on the semi-torus. In this example, the testing error is computed with ℓ_∞ norm over 10,000 uniformly sampled data. Notice that the error of the Nyström is quite large even for $J = 600$. In this case, this large error is attributed to the slow convergence of the estimation of eigenvectors near the boundary (see [76] for the detailed rate). To highlight this issue, we show the error comparison of the Nystrom (J=600) and NN solutions for $N = 32400$ in Figure 4. We can see that the error of Nystrom concentrates near the boundary while the boundary error of the NN solution is small.

Section summary: Based on the numerical comparisons in the above three examples, we found that the NN solution produces more robust and accurate generalizations compared to the Nyström-based interpolation. The

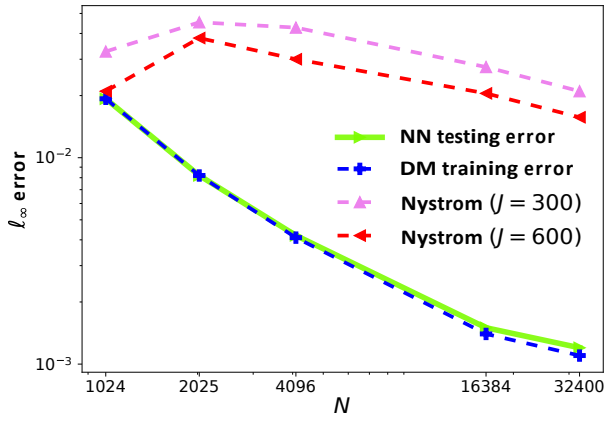
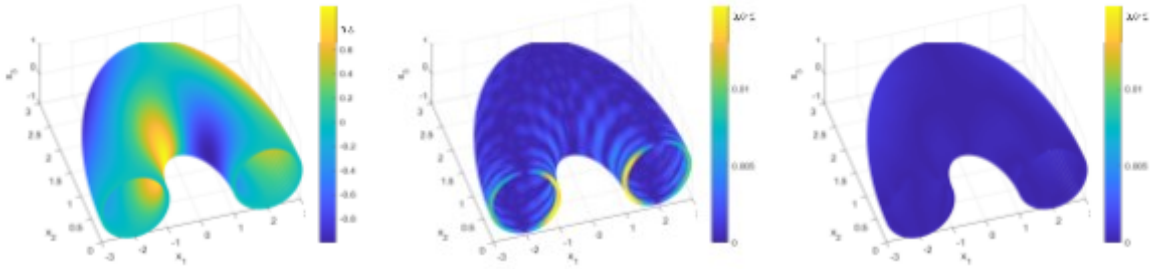


Figure 3: **Semi-torus:** The comparison of the ℓ_∞ errors as functions of the number of training points N for DM and NN on a semitorus embedded in $\mathbf{R}^3$.



(a) True solution (b) Difference between Nystrom & truth (c) Difference between NN & truth

Figure 4: Comparison of the PDE solutions among the Nystrom extension ($J = 600$) and NN on the semi-torus example ($N = 32400$). (a) True solution. (b) Absolute difference between Nystrom and true solutions. (c) Absolute difference NN and true solutions.

Nyström-based method requires a large number of eigenfunctions to suppress the residual for general PDE solution. This requirement translates to the need for a large number of training data for accurate estimation of eigenvectors corresponding to large eigenvalues of the Laplace-Beltrami operator, which also means more computational power is needed in solving the corresponding eigenvalue problems. Based on this finding, we will not present the Nyström-based interpolation in the next three examples. We will use the DM (or VBDM) and NN training errors as benchmarks for the NN testing error.

5.3 A 3D Manifold of High Co-Dimension

In this section, we demonstrate the performance of the NN regression solution on a simple manifold with high co-dimension. For this example, we will also compare results from the Polynomial-Sine to other simpler activation functions. Specifically, we consider the elliptic PDE in (1) with $a = 0, \kappa = 1$ on a closed manifold M, embedded by $\iota : M \to \mathbf{R}^{12}$, defined through the following embedding function,

$$\iota(t_1, t_2, t_3) := \begin{pmatrix} \sin(t_1), \cos(t_1), \sin(2t_1), \cos(2t_1), \sin(t_2), \cos(t_2), \sin(2t_2), \cos(2t_2), \sin(t_3), \cos(t_3), \sin(2t_3), \cos(2t_3) \end{pmatrix},$$

for $t_1, t_2, t_3 \in [0, 2\pi)$. We manufacture the right hand data f by setting the true solution to be $u(t_1, t_2, t_3) = \sin t_1 \cos t_2 \sin 2t_3$. The training points $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ are well sampled (equal angles in the intrinsic coordinates (t_1, t_2, t_3)). We solve the PDE problem by minimizing the loss function (19) with $\lambda = 0$ since the problem has no boundary. The hyperparameter setting of DM and NN is presented in Table 9 in Appendix C.

Figure 5 displays the error for DM and NN, where the latter is the solution we obtained using the Polynomial-Sine activation function. Here, the testing error refers to the ℓ_∞ error on 80^3 Gauss-Legendre quadrature points obtained from the intrinsic coordinates (t_1, t_2, t_3). From Figure 5, one can see that the NN solver produces convergent solutions. Besides, when N is large (e.g., $N = 4096$ or 12167), NN obtains a slightly more accurate solution than DM. We report the training and testing errors for different activation functions and optimizers in Table 2. The training errors of Polynomial-Sine with Adam are comparable to that of ReLU and ReLU^3 with gradient descent, but the Polynomial-Sine FNN optimized by Adam obtains the lowest testing error for most N. This finding is not surprising since the true solution is a product of sine and cosine functions. In the next section, we will consider problems where the true solutions are unknown and we will see that other type of activation functions produce more accurate results.

Activation	Optimizer	N	512	1331	4096	12167	24389
Polynomial-Sine	Adam	training error	0.2665	0.1148	0.0297	0.0066	0.0023
		testing error	0.2715	0.1346	0.0302	0.0069	0.0024
ReLU^3	gradient descent	training error	0.2624	0.1137	0.0309	0.0058	0.0025
		testing error	0.2994	0.1977	0.0425	0.0147	0.0103
ReLU	gradient descent	training error	0.2637	0.1145	0.032	0.0066	0.0016
		testing error	0.4673	0.198	0.0326	0.0069	0.0016

Table 2: The comparison of the ℓ_∞ errors for different activation functions and optimizers on the 3D manifold embedded in $\mathbf{R}^{12}$.

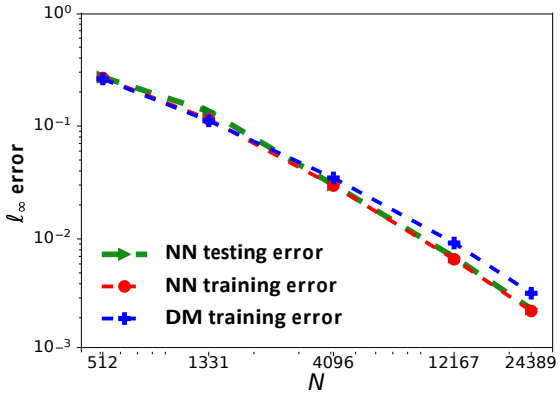


Figure 5: The comparison of the ℓ_∞ errors as functions of the number of training points N for DM and NN on the 3D manifold embedded in $\mathbf{R}^{12}$.

5.4 Unknown surfaces in $\mathbf{R}^3$

In this subsection, we consider solving the elliptic PDEs on the unknown surfaces in $\mathbf{R}^3$ with or without boundary, including the Bunny (no boundary) and Face (with boundary). The data set for the Bunny surface is from the Stanford 3D Scanning Repository [1]. The original data set of the Stanford Bunny comprises a triangle mesh with 34,817 vertices. Since this data set has singular regions at the bottom, we generate a new mesh of the surface using the Marching Cubes algorithm [60] that is available through the Meshlab [71]. We should point out that the Marching Cubes algorithm does not smooth the surface. Instead, we will use the vertices of the new mesh as sample points to avoid singularity induced by the original data set. To generate various sizes of sample points, we subsequently apply the Quadric edge algorithm [35] to simplify the mesh obtained by the Marching Cubes algorithm (which is a surface of 34,594 vertices) into 4,326, 8,650, and 17,299 vertices. The surface Face is obtained from Keenan Crane's 3D repository [18]. The original Face consists of 17,157 vertices and we generate the datasets of 2,185, 4,340, and 8,261 vertices by employing the Quadric edge algorithm [35] in Meshlab [71]. Besides these data resampling, we also multiply the coordinate of Face by 10 to avoid a solution close to zero function. Since the analytic solution is not accessible, we benchmark our solution against the finite element method (FEM) solution, obtained from the FELICITY FEM Matlab toolbox [93]. The estimator $\mathbf{L}_\epsilon$ of the Laplace–Beltrami operator is constructed by the variable bandwidth diffusion mapping (VBDM). When training with less than the largest available data set (34,594 vertices for the Bunny and 17,157 vertices for the Face), we report the testing error, which is based on ℓ_∞ norm over the largest data set. So, the testing errors reported in the following two examples also include the interpolation error since they are the maximum errors of the solutions on both the training and testing data sets.

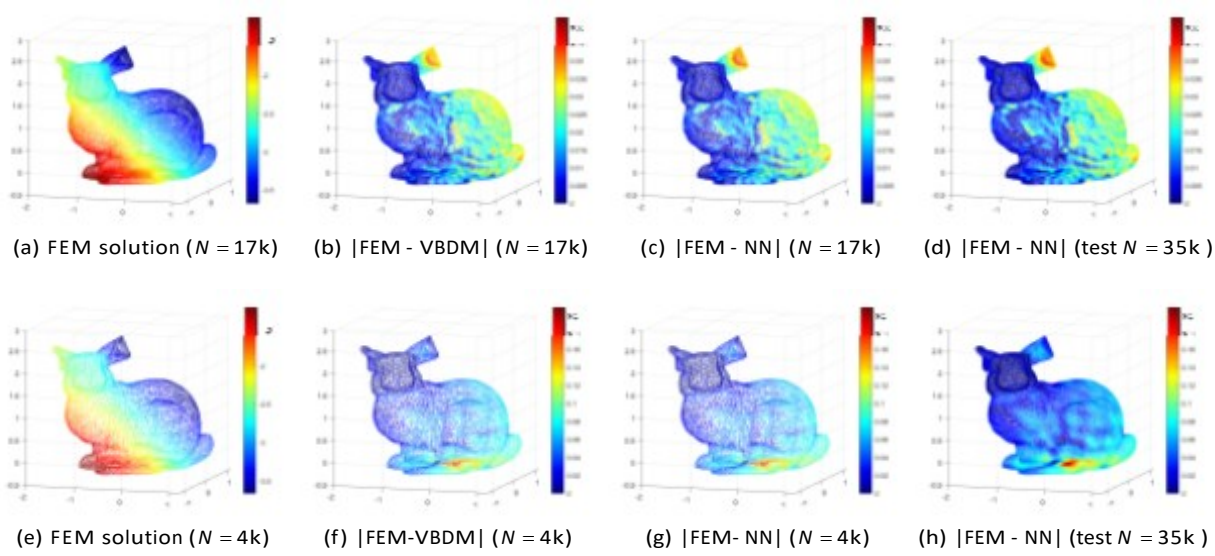
5.4.1 The Stanford Bunny

We denote the point cloud (vertices) of this surface as, $\boldsymbol{x} = (x_1, x_2, x_3) \in M \subset \mathbf{R}^3$. Here, we consider solving the PDE in (1) with $a = 0.2, \kappa = 1$ and $f = x_1 + x_2 + x_3$. We solve the PDE problem by minimizing the loss function (19) with $\gamma = 0$ and $\lambda = 0$ since the problem has nontrivial a and no boundary. The hyperparameter setting of DM and NN is presented in Table 10 in Appendix C.

Table 3 shows the error comparison of the solutions between VBDM and NN methods with the growth of the data point size. We can see that the NN solver gives convergent solutions as VBDM. We should also point out that the similar training error rates of NN and VBDM reported in this table suggest that the error is dominated by the VBDM approximation. The rate, however, is faster than the theoretical error bound $\epsilon \sim N^{-\frac{2}{6+d}}$ attained by balancing the first and third bounds in Theorem 4.1 since the numerical experiment is conducted with empirically chosen parameters ϵ and k in k-nearest neighbors algorithm to optimize the accuracy of the solutions (see Appendix C for the chosen parameters). Besides, the NN solutions suggest a good generalization as the testing error is consistent with its training error. Figure 6 displays the visualization for the case of $N = 17299$ and $N = 4326$ in Table 3.

	N	4326	8650	17299	34594
VBDM	training error	0.2047	0.1017	0.0514	0.0283
NN	training error	0.2048	0.1019	0.0518	0.0296
	testing error	0.2050	0.1023	0.0517	–

Table 3: Error comparison of the PDE solutions between VBDM and NN. The testing error is the ℓ_∞ error between the FEM and the NN solutions on the 34,594 vertices.



(a) FEM solution ($N = 17$k) (b) |FEM - VBDM| ($N = 17$k) (c) |FEM - NN| ($N = 17$k) (d) |FEM - NN| (test $N = 35$k)

(e) FEM solution ($N = 4$k) (f) |FEM-VBDM| ($N = 4$k) (g) |FEM- NN| ($N = 4$k) (h) |FEM - NN| (test $N = 35$k)

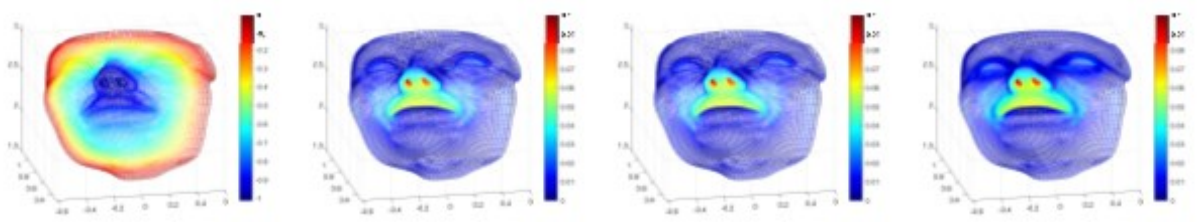
Figure 6: Comparison of the PDE solutions among FEM, VBDM and NN on the Bunny example. **(a)** FEM solution with $N = 17299$ training data. **(b)** Absolute difference between FEM and VBDM solutions on training data set of size $N = 17299$. **(c)** Absolute difference between FEM and NN solutions on training data set of size $N = 17299$. **(d)** Absolute difference between FEM and NN solutions on 34594 data points from the Marching Cubes algorithm, 17299 of these points were used to obtain errors in (b) and (c). **(e)** FEM solution with $N = 4,326$ training data. **(f)** Absolute difference between FEM and VBDM solutions on training data set of size $N = 4,326$. (g) Absolute difference between FEM and NN solutions on training data set of size $N = 4,326$. (h) Exactly like (d) except that only 4326 of these points are used to obtain errors in (f) and (g).

5.4.2 Face example with the Dirichlet boundary condition.

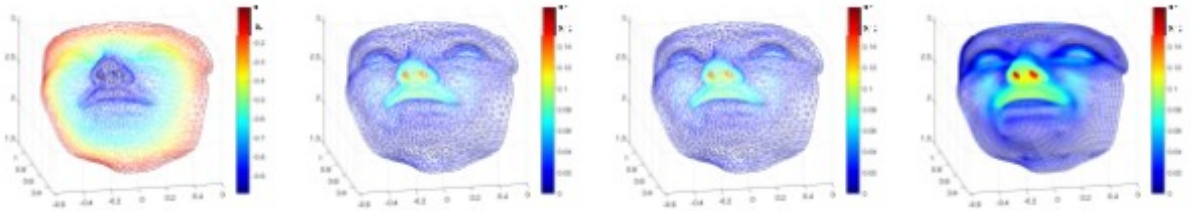
In this face example, we let $a = 0, \kappa = 1$ and $f = x_1 + x_2 + x_3$ in the PDE (1), where $\boldsymbol{x} = (x_1, x_2, x_3) \in M \subset \mathbf{R}^3$, and we impose the Dirichlet boundary condition ($u = 0$ on ∂M). The hyperparameter setting can be referred to Table 11. In this example, we found that the constructed $\mathbf{L}_\epsilon$ is extremely ill-conditioned (the condition number is 2.0×10^5 when $N = 17157$), the least-squares optimization becomes difficult. To facilitate the convergence of NN training, we increase the training iteration and use a more time-efficient activation function (such as ReLU and Tanh) instead of Polynomial-Sine function. Table 4 shows the error comparison of the solutions between VBDM and NN methods with the growth of the data point size. We can see that the NN solver gives the solutions with error similar to VBDM on the training data points. Besides, the testing error of NN is close to its training error. Figure 7 displays the visualization of the FEM, VBDM and NN solutions.

	N	2185	4340	8261	17157
VBDM	training error	0.1548	0.1340	0.0967	0.0708
NN	training error	0.1548	0.1340	0.0967	0.0911
	testing error	0.1761	0.1429	0.0984	–

Table 4: Error comparison of the PDE solutions between VBDM and NN. The testing error is the ℓ_∞ error between the FEM and the NN solutions on the 17,157 vertices.



(a) FEM solution ($N = 8$k) (b) |FEM - VBDM| ($N = 8$k) (c) |FEM - NN| ($N = 8$k) (d) |FEM - NN| (test $N = 17$k)



(e) FEM solution ($N = 2$k) (f) |FEM - VBDM| ($N = 2$k) (g) |FEM - NN| ($N = 2$k) (h) |FEM - NN| (test $N = 17$k)

Figure 7: Comparison of the PDE solutions among FEM, VBDM and NN on the Bunny example. **(a)** FEM solution with $N = 8261$ training data. **(b)** Absolute difference between FEM and VBDM solutions on training data set of size $N = 8261$. **(c)** Absolute difference between FEM and NN solutions on training data set of size $N = 8261$. **(d)** Absolute difference between FEM and NN solutions on 17157 data points from the Marching Cubes algorithm, 8261 of these points were used to obtain errors in (b) and (c). **(e)** FEM solution with $N = 2185$ training data. **(f)** Absolute difference between FEM and VBDM solutions on training data set of size $N = 2185$. (g) Absolute difference between FEM and NN solutions on training data set of size $N = 2185$. (h) Exactly like (d) except that only 2185 of these points are used to obtain errors in (f) and (g).

6 Conclusion

This paper proposed a mesh-free computational framework and machine learning theory for solving PDEs on unknown manifolds given as a form of point clouds based on diffusion maps (DM) and deep learning. Parameterizing manifolds is challenging if the unknown manifold is embedded in a high-dimensional ambient Euclidean space, especially when the manifold is identified with randomly sampled data and has boundaries. First, a mesh-free DM algorithm was introduced to approximate differential operators on point clouds enabling the design of PDE solvers on manifolds with and without boundaries. Second, deep NNs were applied to parametrize PDE solutions. Finally, we solved a least-squares minimization problem for PDEs, where the empirical loss function is evaluated using the DM discretized differential operators on point clouds and the minimizer is identified via SGD. The minimizer provides a PDE solution in the form of an NN function on the whole unknown manifold with reasonably good accuracy. The mesh-free nature and randomization of the proposed solver enable efficient solutions to PDEs on manifolds arbitrary co-dimensional. Computationally, we found that training NN solutions are computationally more feasible compared to performing (a stable) pseudo-inversion in solving linear problems with a singular matrix as the size of the matrix becomes large. On the other hand, this advantage diminishes when the linear problem involves inversion of non-singular matrix and the goal is to achieve high accuracy; where classical algorithms (e.g., GMRES, iterative methods) are more efficient. When the linear system is too large to compute, it was recently pointed out in [38] that there are advantages to apply NNs to solve linear systems in a similar way as in this paper even though the linear system is non-singular, while traditional methods are not affordable. Beyond the training

procedure, the NN solution is advantageous for interpolating the solution to a new datum. From the numerical results, we found that the proposed approach produces consistently accurate generalizations that are more robust than the one constructed using the classical Nyström interpolation technique, especially when the latter is practically not convenient due to the following fact. The accuracy of the Nyström extension depends crucially on the choice of the dimension of the eigenspace, J, that allow for the projected solution in (20) to have a small residual in addition to estimating the leading J eigenvector $\mathbf{v}_i$ of Laplace-Beltrami operator.

New convergence and consistency theories based on approximation and optimization analysis were developed to support the proposed framework. From the perspective of algorithm development, it is interesting to extend the proposed framework to vector-valued PDEs with other differential operators in future work. In terms of theoretical analysis, it is important to develop a generalization analysis of the proposed solver in the future and extend all of the analysis in this paper to manifolds with boundaries.

Code availability

The neural network training codes are available at the GitHub repository: https://github.com/LeungSamWai/NN4ManifoldPDE. The source codes are released under MIT license.

Acknowledgment

The research of J. H. was partially supported under the NSF grant DMS-1854299 and the ONR grant N00014-22-1-2193. S. J. was supported by the NSFC Grant No. 12101408. S. L. is supported by the Ross-Lynn fellowship 2021-2022 from Purdue University. S. L. and H. Y. were partially supported by the US National Science Foundation under awards DMS-2244988, DMS-2206333, and the Office of Naval Research Award N00014-23-1-2007.

A GPDM algorithm for manifolds with boundaries

In this appendix, we give a brief overview of ghost point diffusion maps (GPDM) to construct the matrix $\mathbf{L}_\epsilon$ for manifolds with Dirichlet boundary conditions. As mentioned in [16, 44, 36, 51], the asymptotic expansion (2) in standard DM approaches is not valid near the boundary of the manifold. One way to overcome this boundary issue is the GPDM approach introduced in [51]. This approach extends the classical ghost point method [55] to solve elliptic and parabolic PDEs on unknown manifolds with boundary conditions [51, 94]. The GPDM approach can be summarized as follows (see [51, 94] for details).

Estimation of normal vectors at boundary points (see details in Section 2.2 and Appendix A of [94])**:** Assume the true normal vector $\boldsymbol{\nu}$ is unknown and it will be numerically estimated. For well-sampled data, where data points are well-ordered along intrinsic coordinates, one can identify the estimated $\boldsymbol{\nu}$ as the secant line approximation to $\boldsymbol{\nu}$. The error is $|\boldsymbol{\nu} - \tilde{\boldsymbol{\nu}}| = O(h)$, where the parameter h denotes the distance between consecutive ghost points (see Fig. 1(b) in [94] for a geometric illustration). For randomly sampled data, one can use the kernel method to estimate $\tilde{\boldsymbol{\nu}}$ and the error is $|\boldsymbol{\nu} - \tilde{\boldsymbol{\nu}}| = O(\epsilon^{\frac{1}{4}})$ (see Fig. 1(c) in [94] for a geometric illustration and Appendix A in [94] for a detailed discussion).

Specification of ghost points: The basic idea of ghost points, as introduced in [51], is to specify the ghost points as data points that lie on the exterior normal collar, ΔM, along the boundary. Then all interior points whose distances are within ϵ^r from the boundary ∂M are at least ϵ^r away from the boundary of the extended manifold $M \cup \Delta M$. Theoretically, it was shown that, under appropriate conditions, the extended set $M \cup \Delta M$ can be isometrically embedded with an embedding function that is consistent with the embedding $M \hookrightarrow \mathbf{R}^m$ when restricted on M (see Lemma 3.5 in [51]).

Technically, for randomly sampled data, the parameter h can be estimated by the mean distance from the boundary $\mathbf{x}_b$ to its P (around 10 in simulations) nearest neighbors. Then, given the distance parameter h and the estimated normal vector $\tilde{\boldsymbol{\nu}}$, the approximate ghost points are given by,

$$\tilde{\mathbf{x}}_{b,k} = \mathbf{x}_b + k\,h\,\tilde{\boldsymbol{\nu}}, \quad \text{for } k = 1, \ldots, K \text{ and } b = 1, \ldots, N_b. \tag{24}$$

In addition, one layer of interior ghost points are supplemented as $\tilde{\mathbf{x}}_{b,0} = \mathbf{x}_b - h\tilde{\boldsymbol{\nu}}$. For well-sampled data, the interior estimated ghost point coincides with one of the interior points on the manifold when the secant line method is used. However, for randomly sampled data, the estimated interior ghost points will not necessarily coincide with an interior point (see Fig. 1 in [94] for comparison).

Estimation of function values on the ghost points:

The main goal here is to estimate the function values $\{u(\tilde{\mathbf{x}}_{b,k})\}_{b,k=1}^{N_b,K}$ on the exterior ghost points by extrapolation. We assume that we are given the components of the column vector,

$$\mathbf{u}_M := (u(\mathbf{x}_1), \ldots, u(\tilde{\mathbf{x}}_{b,0}), \ldots, u(\mathbf{x}_N)) \in \mathbf{R}^N. \tag{25}$$

Here, we stress that the function values $\{u(\tilde{\mathbf{x}}_{b,0})\}_{b=1}^{N_b}$ are given exactly like the $u(\mathbf{x}_i)$ for any $\mathbf{x}_i \in M$, even when the ghost points $\tilde{\mathbf{x}}_{b,0}$ do not lie on the manifold M. Then we will use the components of the column vector,

$$\mathbf{U}_G := (U_{1,1}, \ldots, U_{N_b,K}) \in \mathbf{R}^{N_b K}, \tag{26}$$

to estimate the components of function values on ghost points, $(u(\tilde{\mathbf{x}}_{1,1}), \ldots, u(\tilde{\mathbf{x}}_{N_b,K})) \in \mathbf{R}^{N_b K}$. Numerically, we will obtain the components of $\mathbf{U}_G$ by solving the following linear algebraic equations for each $b = 1, \ldots, N_b$,

$$\begin{aligned} U_{b,1} - 2u(\mathbf{x}_b) + u(\tilde{\mathbf{x}}_{b,0}) &= 0, \\ U_{b,2} - 2U_{b,1} + u(\mathbf{x}_b) &= 0, \\ U_{b,k} - 2U_{b,k-1} + U_{b,k-2} &= 0, \qquad k = 3, \ldots K. \end{aligned} \tag{27}$$

These algebraic equations are discrete analogs of matching the first-order derivatives along the estimated normal direction, $\tilde{\boldsymbol{\nu}}$.

Construction of the GPDM estimator: We now define the GPDM estimator for the differential operator L in (1). The discrete estimator will be constructed based on the available training data $\{\mathbf{x}_i \in M\}_{i=1}^N$ and the estimated ghost points, $\{\tilde{\mathbf{x}}_{b,k}\}_{b,k=1,0}^{N_b,K}$. In particular, since the interior ghost points, $\{\tilde{\mathbf{x}}_{b,0}\}_{b=1}^{N_b}$ may or may not coincide with any interior points on the manifold, we assume that $X^h := \{\mathbf{x}_1, \ldots, \tilde{\mathbf{x}}_{b,0}, \ldots, \mathbf{x}_N\}$ has N components that include the estimated ghost points in the following discussion. With this notation, we define a non-square matrix,

$$\mathbf{L}^h := (\mathbf{L}^{(1)}, \mathbf{L}^{(2)}) \in \mathbf{R}^{N \times (N + N_b K)}, \tag{28}$$

constructed right after (3), by evaluating the kernel on components of X^h for each row and the components of $X^h \cup \{\tilde{\mathbf{x}}_{b,k}\}_{b,k=1}^{N_b,K}$ for each column. With these definitions and those in (25)-(27), we note that

$$\mathbf{L}^h \mathbf{U} = \mathbf{L}^{(1)} \mathbf{u}_M + \mathbf{L}^{(2)} \mathbf{U}_G = (\mathbf{L}^{(1)} + \mathbf{L}^{(2)} \mathbf{G}) \mathbf{u}_M = \tilde{\mathbf{L}} \mathbf{u}_M,$$

where $\mathbf{U} := (\mathbf{u}_M, \mathbf{U}_G) \in \mathbf{R}^{N + N_b K}$ and $\mathbf{G} \in \mathbf{R}^{N_b K \times N}$ is defined as a solution operator to (27), which is given in a compact form as $\mathbf{U}_G = \mathbf{G} \mathbf{u}_M$. Then, the GPDM estimator $\tilde{\mathbf{L}}$ is defined as an $N \times N$ matrix,

$$\tilde{\mathbf{L}} := \mathbf{L}^{(1)} + \mathbf{L}^{(2)} \mathbf{G}. \tag{29}$$

Note that we have the consistency of GPDM estimator for the differential operator defined on functions that take values on the extended $M \cup \Delta M$ (see Lemma 2.3 in [94]).

Combination with the discretization of the boundary conditions: We take $\mathbf{L}_\epsilon \in \mathbf{R}^{(N - N_b) \times N}$ to be a sub-matrix of $\tilde{\mathbf{L}} \in \mathbf{R}^{N \times N}$ in (29), where the $N - N_b$ rows of $\mathbf{L}_\epsilon$ correspond to the interior points from X^h. There are $N - N_b$ equations from the linear system $(-\mathbf{a} + \mathbf{L}_\epsilon)\mathbf{u}_M = \mathbf{f}$. To close the discretized problem, we use the N_b equations from the Dirichlet boundary condition at the boundary points, $u(\bar{\mathbf{x}}_b) = g(\bar{\mathbf{x}}_b)$ for $\bar{\mathbf{x}}_b \in \{\bar{\mathbf{x}}_1, \ldots, \bar{\mathbf{x}}_{N_b}\} \subset X^h \cap \partial M$. With the construction of $\mathbf{L}_\epsilon$ and the Dirichlet boundary conditions, we then solve the problem in (9) using DNN approach.

B Global convergence analysis of neural network optimization

In this section, we will summarize notations and main ideas of the proof of Theorem 4.2 first. Then we prove several lemmas in preparation for the proof of Theorem 4.2. The proof of Theorem 4.2 will be presented after these lemmas.

B.1 Preliminaries

The Rademacher complexity is a basic tool for generalization analysis. In our analysis, we will use several important lemmas and theorems related to it. To be self-contained, they are listed as follows.

Definition B.1 (The Rademacher complexity of a function class $\mathcal{F}$)**.** *Given a sample set $S = \{z_1, \ldots, z_N\}$ on a domain $\mathcal{Z}$, and a class $\mathcal{F}$ of real-valued functions defined on $\mathcal{Z}$, the empirical Rademacher complexity of $\mathcal{F}$ on S is defined as*

$$\mathrm{Rad}_S(\mathcal{F}) = \frac{1}{N} \mathbf{E}_{\boldsymbol{\tau}} \left[\sup_{f \in \mathcal{F}} \sum_{i=1}^N \tau_i f(z_i) \right],$$

where $\tau_1, \ldots, \tau_N$ are independent random variables drawn from the Rademacher distribution, i.e., $\mathbf{P}(\tau_i = +1) = \mathbf{P}(\tau_i = -1) = \frac{1}{2}$ for $i = 1, \ldots, N$.

First, we recall a well-known contraction lemma for the Rademacher complexity.

Lemma B.1 (Contraction lemma [82]). *Suppose that $\psi_i : \mathbf{R} \to \mathbf{R}$ is a C_L-Lipschitz function for each $i \in [N] := \{1, \dots, N\}$. For any $\boldsymbol{y} \in \mathbf{R}^N$, let $\boldsymbol{\psi}(\boldsymbol{y}) = (\psi_1(y_1), \bullet\bullet\bullet, \psi_N(y_N))^\top$. For an arbitrary set of functions F on an arbitrary domain Z and an arbitrary choice of samples $S = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_N\} \subset \mathsf{Z}$, we have*

$$\mathrm{Rad}_S(\boldsymbol{\psi} \circ \mathsf{F}) \le C_L \mathrm{Rad}_S(\mathsf{F}).$$

Second, the Rademacher complexity of linear predictors can be characterized by the lemma below.

Lemma B.2 (Rademacher complexity for linear predictors [82]). *Let $\Theta = \{\boldsymbol{w}_1, \bullet\bullet\bullet, \boldsymbol{w}_m\} \in \mathbf{R}^n$. Let $\mathsf{G} = \{g(\boldsymbol{w}) = \boldsymbol{w}^\top \boldsymbol{x} : \|\boldsymbol{x}\|_1 \le 1\}$ be the linear function class with parameter $\boldsymbol{x}$ whose ℓ^1 norm is bounded by 1. Then*

$$\mathrm{Rad}_\Theta(\mathsf{G}) \le \max_{1 \le k \le m} \|\boldsymbol{w}_k\|_\infty \sqrt{\frac{2\log(2n)}{m}}.$$

Finally, let us state a general theorem concerning the Rademacher complexity and generalization gap of an arbitrary set of functions F on an arbitrary domain Z, which is essentially given in [82].

Theorem B.1 (Rademacher complexity and generalization gap [82]). *Suppose that f's in F are non-negative and uniformly bounded, i.e., for any $f \in \mathsf{F}$ and any $\boldsymbol{z} \in \mathsf{Z}$, $0 \le f(\boldsymbol{z}) \le B$. Then for any $\delta \in (0,1)$, with probability at least $1 - \delta$ over the choice of N i.i.d. random samples $S = \{\boldsymbol{z}_1, \dots, \boldsymbol{z}_N\} \subset \mathsf{Z}$, we have*

$$\sup_{f \in \mathsf{F}} \left| \frac{1}{N} \sum_{i=1}^N f(\boldsymbol{z}_i) - \mathbf{E}_{\boldsymbol{z}} f(\boldsymbol{z}) \right| \le 2\mathbf{E}_S \mathrm{Rad}_S(\mathsf{F}) + B\sqrt{\frac{\log(2/\delta)}{2N}},$$
$$\sup_{f \in \mathsf{F}} \left| \frac{1}{N} \sum_{i=1}^N f(\boldsymbol{z}_i) - \mathbf{E}_{\boldsymbol{z}} f(\boldsymbol{z}) \right| \le 2\mathrm{Rad}_S(\mathsf{F}) + 3B\sqrt{\frac{\log(4/\delta)}{2N}}.$$

Here, we should point out that the distribution of $\boldsymbol{z}$ is arbitrary. In our specific application, $\mathbf{E}_{\boldsymbol{z}} = \mathbf{E}_\pi$.

B.2 Notations and main ideas of the proof of Theorem 4.2

Now we are going to present the notations and main ideas of the proof of Theorem 4.2. Recall that we use a two-layer neural network $\varphi(\boldsymbol{x}; \boldsymbol{\vartheta})$ with $\boldsymbol{\vartheta} = \mathrm{vec}\{a_k, \boldsymbol{w}_k\}_{k=1}^m$. In the gradient descent iteration, we use t to denote the iteration or the artificial time variable in the gradient flow. Hence, we define the following notations for the evolution of parameters at time t:

$$a_k^t := a_k(t), \quad \boldsymbol{w}_k^t := \boldsymbol{w}_k(t), \quad \boldsymbol{\vartheta}^t := \boldsymbol{\vartheta}(t) := \mathrm{vec}\{a_k^t, \boldsymbol{w}_k^t\}_{k=1}^m.$$

Similarly, we can introduce t to other functions or variables depending on $\boldsymbol{\vartheta}(t)$. When the dependency of t is clear, we will drop the index t. In the initialization of gradient descent, we set

$$\begin{aligned} &a_k^0 := a_k(0) \sim \mathrm{N}(0, \gamma^2), \quad \boldsymbol{w}_k^0 := \boldsymbol{w}_k(0) \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{I}_n), \\ &\boldsymbol{\vartheta}^0 := \boldsymbol{\vartheta}(0) := \mathrm{vec}\{a_k^0, \boldsymbol{w}_k^0\}_{k=1}^m. \end{aligned} \tag{30}$$

Then the empirical risk can be written as $R_S(\boldsymbol{\vartheta}) = \frac{1}{2N} \boldsymbol{e}^\top \boldsymbol{A}^\top \boldsymbol{A} \boldsymbol{e}$, where we denote $e_i = \varphi(\boldsymbol{x}_i; \boldsymbol{\vartheta}) - (\boldsymbol{A}^{-1} f(X))_i$ for $i \in [N]$ and $\boldsymbol{e} = (e_1, e_2, \dots, e_N)^\top$. Hence, the gradient descent dynamics is $\dot{\boldsymbol{\vartheta}} = -\nabla_{\boldsymbol{\vartheta}} R_S(\boldsymbol{\vartheta})$, or equivalently in terms of a_k and $\boldsymbol{w}_k$ as follows:

$$\begin{aligned} \dot{a}_k &= -\nabla_{a_k} R_S(\boldsymbol{\vartheta}) = -\frac{1}{N} \sum_{i=1}^N (\boldsymbol{e}^\top \boldsymbol{A}^\top \boldsymbol{A})_i \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}_i), \\ \dot{\boldsymbol{w}}_k &= -\nabla_{\boldsymbol{w}_k} R_S(\boldsymbol{\vartheta}) = -\frac{1}{N} \sum_{i=1}^N (\boldsymbol{e}^\top \boldsymbol{A}^\top \boldsymbol{A})_i a_k \sigma'(\boldsymbol{w}_k^\top \boldsymbol{x}_i) \boldsymbol{x}_i. \end{aligned} \tag{31}$$

Adopting the neuron tangent kernel point of view [50], in the case of a two-layer neural network with an infinite width, the corresponding kernels $k^{(a)}$ for parameters in the last linear transform and $k^{(w)}$ for parameters in the first layer are functions from $M \times M$ to $\mathbf{R}$ defined by

$$\begin{aligned} k^{(a)}(\boldsymbol{x}, \boldsymbol{x}') &:= \mathbf{E}_{\boldsymbol{w} \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{I}_n)} g^{(a)}(\boldsymbol{w}; \boldsymbol{x}, \boldsymbol{x}'), \\ k^{(w)}(\boldsymbol{x}, \boldsymbol{x}') &:= \mathbf{E}_{(a, \boldsymbol{w}) \sim \mathrm{N}(\boldsymbol{0}, \boldsymbol{I}_{n+1})} g^{(w)}(a, \boldsymbol{w}; \boldsymbol{x}, \boldsymbol{x}'), \end{aligned}$$

where

$$g^{(a)}(\boldsymbol{w};\boldsymbol{x},\boldsymbol{x}') := \left[\sigma(\boldsymbol{w}^{\top}\boldsymbol{x})\right]\left[\sigma(\boldsymbol{w}^{\top}\boldsymbol{x}')\right],$$
$$g^{(w)}(a,\boldsymbol{w};\boldsymbol{x},\boldsymbol{x}') := a^2\left[\sigma'(\boldsymbol{w}^{\top}\boldsymbol{x})\boldsymbol{x}\right]\left[\sigma'(\boldsymbol{w}^{\top}\boldsymbol{x}')\boldsymbol{x}'\right].$$

These kernels evaluated at $N \times N$ pairs of samples lead to $N \times N$ Gram matrices $\boldsymbol{K}^{(a)}$ and $\boldsymbol{K}^{(w)}$ with $K^{(a)}_{ij} = k^{(a)}(\boldsymbol{x}_i,\boldsymbol{x}_j)$ and $K^{(w)}_{ij} = k^{(w)}(\boldsymbol{x}_i,\boldsymbol{x}_j)$, respectively. Our analysis requires the matrix $\boldsymbol{K}^{(a)}$ to be positive definite, which has been verified for regression problems under mild conditions on random training data $X = \{\boldsymbol{x}_i\}_{i=1}^N$ and can be generalized to our case. Hence, we assume this together with the non-singularity of $\boldsymbol{A}$.

For a two-layer neural network with m neurons, define the $N \times N$ Gram matrix $\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta})$ and $\boldsymbol{G}^{(w)}(\boldsymbol{\vartheta})$ using the following expressions for the (i,j)-th entry

$$\begin{aligned}\boldsymbol{G}^{(a)}_{ij}(\boldsymbol{\vartheta}) &:= \frac{1}{m}\sum_{k=1}^{m} g^{(a)}(\boldsymbol{w}_k;\boldsymbol{x}_i,\boldsymbol{x}_j),\\ \boldsymbol{G}^{(w)}_{ij}(\boldsymbol{\vartheta}) &:= \frac{1}{m}\sum_{k=1}^{m} g^{(w)}(a_k,\boldsymbol{w}_k;\boldsymbol{x}_i,\boldsymbol{x}_j).\end{aligned} \tag{32}$$

Clearly, $\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta})$ and $\boldsymbol{G}^{(w)}(\boldsymbol{\vartheta})$ are both positive semi-definite for any $\boldsymbol{\vartheta}$. Let $\boldsymbol{G}(\boldsymbol{\vartheta}) = \boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}) + \boldsymbol{G}^{(w)}(\boldsymbol{\vartheta})$, taking time derivative of (18) and using the equalities in (31) and (32), then we have the following evolution equation to understand the dynamics of the gradient descent method applied to (18):

$$\begin{aligned}\frac{\mathrm{d}}{\mathrm{d}t}R_S(\boldsymbol{\vartheta}) = -\left\|\nabla_{\boldsymbol{\vartheta}}R_S(\boldsymbol{\vartheta})\right\|_2^2 &= -\frac{m}{N^2}\boldsymbol{e}^{\top}\boldsymbol{A}^{\top}\boldsymbol{A}\boldsymbol{G}(\boldsymbol{\vartheta})\boldsymbol{A}^{\top}\boldsymbol{A}\boldsymbol{e}\\ &\le -\frac{m}{N^2}\boldsymbol{e}^{\top}\boldsymbol{A}^{\top}\boldsymbol{A}\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta})\boldsymbol{A}^{\top}\boldsymbol{A}\boldsymbol{e}.\end{aligned} \tag{33}$$

Our goal is to show that $R_S(\boldsymbol{\vartheta})$ converges to zero. These goals are true if the smallest eigenvalue $\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta})\right)$ of $\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta})$ has a positive lower bound uniformly in t, since in this case we can solve (33) and bound $R_S(\boldsymbol{\vartheta})$ with a function in t converging to zero when $t \to \infty$ as shown in Lemma B.6. In fact, a uniform lower bound of $\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta})\right)$ can be $\frac{1}{2}\lambda_S$, which can be proved in the following three steps:

- (**Initial phase**) By Assumption 4.1 of $\boldsymbol{K}^{(a)}$, we can show that $\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(0))\right) \approx \lambda_S$ in Lemma B.5 using the observation that $\boldsymbol{K}^{(a)}_{ij}$ is the mean of $g(\boldsymbol{w};\boldsymbol{x}_i,\boldsymbol{x}_j)$ over the normal random variable $\boldsymbol{w}$, while $\boldsymbol{G}^{(a)}_{ij}(\boldsymbol{\vartheta}(0))$ is the mean of $g(\boldsymbol{w};\boldsymbol{x}_i,\boldsymbol{x}_j)$ with m independent realizations.
- (**Evolution phase**) The GD dynamics results in $\boldsymbol{\vartheta}(t) \approx \boldsymbol{\vartheta}(0)$ when over-parametrized as shown in Lemma B.7, meaning that $\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(0))\right) \approx \lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(t))\right)$.
- (**Final phase**) To show the uniform bound $\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(t))\right) \ge \frac{1}{2}\lambda_S$ for all $t \ge 0$, we introduce a stopping time t_0 via

$$t_0 = \inf\{t \mid \boldsymbol{\vartheta}(t) \notin \mathrm{M}(\boldsymbol{\vartheta}^0)\}, \tag{34}$$

where $\mathrm{M}(\boldsymbol{\vartheta}^0) := \left\{\boldsymbol{\vartheta} \mid \left\|\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}) - \boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)\right\|_F \le \frac{1}{4}\lambda_S\right\}$, and show that t_0 is equal to infinity in the final proof of Theorem 4.2.

B.3 Important lemmas for Theorem 4.2

Now we are going to prove several lemmas for Theorem 4.2. In the analysis below, we use $\bar{a}_k^t := \bar{a}_k(t) := \gamma^{-1}a_k(t)$ with $0 < \gamma < 1$, e.g., $\gamma = \frac{1}{\sqrt{m}}$ or $\gamma = \frac{1}{m}$, and $\bar{\boldsymbol{\vartheta}}(t) := \mathrm{vec}\{\bar{a}_k^t, \boldsymbol{w}_k^t\}_{k=1}^m$.

Lemma B.3. *For any $\delta \in (0,1)$ with probability at least $1-\delta$ over the random initialization in* (30)*, we have*

$$\begin{aligned}\max_{k\in[m]}\left\{|a_k^0|, \|\boldsymbol{w}_k^0\|_\infty\right\} &\le \sqrt{2\log\frac{2m(n+1)}{\delta}},\\ \max_{k\in[m]}\left\{|\bar{a}_k^0|\right\} &\le \gamma\sqrt{2\log\frac{2m(n+1)}{\delta}}.\end{aligned} \tag{35}$$

Proof. If $X \sim N(0,1)$, then $\mathbf{P}(|X| > \varepsilon) \le 2e^{-\frac{1}{2}\varepsilon^2}$ for all $\varepsilon > 0$. Since $a_k^0 \sim N(0,1)$, $(\boldsymbol{w}^0)_{k\alpha} \sim N(0,1)$ for $k \in [m], \alpha \in [n]$, and they are all independent, by setting

$$\varepsilon = \sqrt{2\log\frac{2m(n+1)}{\delta}},$$

one can obtain

$$\begin{aligned}
\mathbf{P}\left(\max_{k\in[m]} |a_k^0|, \|\boldsymbol{w}_k^0\|_\infty > \varepsilon\right) &= \mathbf{P}\left(\bigcup_{k\in[m]} |a_k^0| > \varepsilon \cup \bigcup_{k\in[m],\alpha\in[n]} |(\boldsymbol{w}_k^0)_\alpha| > \varepsilon\right) \\
&\le \sum_{k=1}^m \mathbf{P}\left(|a_k^0| > \varepsilon\right) + \sum_{k=1}^m \sum_{\alpha=1}^n \mathbf{P}\left(|(\boldsymbol{w}_k^0)_\alpha| > \varepsilon\right) \\
&\le 2me^{-\frac{1}{2}\varepsilon^2} + 2mne^{-\frac{1}{2}\varepsilon^2} \\
&= 2m(n+1)e^{-\frac{1}{2}\varepsilon^2} \\
&= \delta,
\end{aligned}$$

which implies the conclusions of this lemma. □

Lemma B.4. *For any $\delta \in (0,1)$ with probability at least $1-\delta$ over the random initialization in* (30)*, we have*

$$R_S(\boldsymbol{\vartheta}^0) \le \frac{1}{2}\left(1 + 3\gamma n^r\sqrt{\pi m}\|\boldsymbol{A}\|_2\sqrt{2\log\frac{4m(n+1)}{\delta}}\right)^{(r+1)/2}\left(\sqrt{\frac{r}{2\log(2n)}} + \sqrt{\frac{\log(8/\delta)/2}{r}}\right)^2,$$

Proof. From Lemma B.3 we know that with probability at least $1-\delta/2$,

$$|a_k^0| \le \sqrt{2\log\frac{4m(n+1)}{\delta}} \quad \text{and} \quad \|\boldsymbol{w}_k^0\|_1 \le n\sqrt{2\log\frac{4m(n+1)}{\delta}}.$$

Let

$$\mathcal{H} = \{h(a,\boldsymbol{w};\boldsymbol{x}) \mid h(a,\boldsymbol{w};\boldsymbol{x}) = a\sigma(\boldsymbol{w}^\top\boldsymbol{x}), \boldsymbol{x} \in \Omega\}.$$

Each element in the above set is a function of a and $\boldsymbol{w}$ while $\boldsymbol{x} \in [0,1]^n$ is a parameter. Since $\|\boldsymbol{x}\|_\infty \le 1$, we have

$$|h(a_k^0, \boldsymbol{w}_k^0; \boldsymbol{x})| \le |a_k^0|\|\boldsymbol{w}_k^0\|_1^r \le n^r\left(2\log\frac{4m(n+1)}{\delta}\right)^{(r+1)/2}.$$

Then with probability at least $1-\delta/2$, by the Rademacher-based uniform convergence theorem, we have

$$\begin{aligned}
\frac{1}{\gamma m}\sup_{\boldsymbol{x}\in\Omega}|\varphi(\boldsymbol{x};\boldsymbol{\vartheta}^0)| &= \sup_{\boldsymbol{x}\in\Omega}\left|\frac{1}{m}\sum_{k=1}^m h(a_k^0, \boldsymbol{w}_k^0; \boldsymbol{x}) - \mathbf{E}_{(a,\boldsymbol{w})\sim N(0,I_{n+1})} h(a,\boldsymbol{w};\boldsymbol{x})\right| \\
&\le 2\mathrm{Rad}_{\boldsymbol{\vartheta}^0}(\mathcal{H}) + 3n^r\left(2\log\frac{4m(n+1)}{\delta}\right)^{(r+1)/2}\sqrt{\frac{\log(8/\delta)}{2m}},
\end{aligned}$$

where

$$\mathrm{Rad}_{\boldsymbol{\vartheta}^0}(\mathcal{H}) := \frac{1}{m}\mathbf{E}_{\boldsymbol{\tau}}\left[\sup_{\boldsymbol{x}\in\Omega}\sum_{k=1}^m \tau_k h(a_k^0, \boldsymbol{w}_k^0; \boldsymbol{x})\right] = \frac{1}{m}\mathbf{E}_{\boldsymbol{\tau}}\left[\sup_{\boldsymbol{x}\in\Omega}\sum_{k=1}^m \tau_k a_k^0\sigma(\boldsymbol{w}_k^{0\top}\boldsymbol{x})\right],$$

where $\boldsymbol{\tau}$ is a random vector in $\mathbf{R}^m$ with i.i.d. entries $\{\tau_k\}_{k=1}^m$ following the Rademacher distribution. With probability at least $1-\delta/2$, $\psi_k(y_k) = a_k\sigma(y_k)$ for $k \in [m]$ is a Lipschitz continuous function with a Lipschitz constant

$$rn^{r-1}\left(2\log\frac{4m(n+1)}{\delta}\right)^{1/2}$$

when $y_k \in [-n\sqrt{2\log(4m(n+1)/\delta)}, n\sqrt{2\log(4m(n+1)/\delta)}]$. We continuously extend $\psi_k(y_k)$ to the domain $\mathbf{R}$ with the same Lipschitz constant.

Applying Lemma B.1 with $\psi_k(y_k)$, we have

$$\begin{aligned}
\frac{1}{m}\mathbf{E}_{\boldsymbol{\tau}}\left[\sup_{\boldsymbol{x}\in\Omega}\sum_{k=1}^m \tau_k a_k^0\sigma(\boldsymbol{w}_k^{0\top}\boldsymbol{x})\right] &\le \frac{1}{m}rn^{r-1}\left(2\log\frac{4m(n+1)}{\delta}\right)^{1/2}\mathbf{E}_{\boldsymbol{\tau}}\left[\sup_{\boldsymbol{x}\in\Omega}\sum_{k=1}^m \tau_k\boldsymbol{w}_k^{0\top}\boldsymbol{x}\right] \\
&\le \frac{rn^r\sqrt{2\log(2n)}}{\sqrt{\pi m}}\left(2\log\frac{4m(n+1)}{\delta}\right)^{(r+1)/2}, \qquad (36)
\end{aligned}$$

where the second inequality is by the Rademacher bound for linear predictors in Lemma B.2.

So one can get

$$\sup_{\boldsymbol{x}\in\Omega}|\varphi(\boldsymbol{x};\boldsymbol{\vartheta}^0)| \le \gamma\sqrt{m}n^r\left(2\log\frac{4m(n+1)}{\delta}\right)^{(r+1)/2}\left(2r\sqrt{2\log(2n)}+3\sqrt{\log(8/\delta)/2}\right).$$
$$\le 3\gamma\sqrt{m}n^r\left(2\log\frac{4m(n+1)}{\delta}\right)^{(r+1)/2}\left(r\sqrt{2\log(2n)}+\sqrt{\log(8/\delta)/2}\right).$$

Then

$$R_S(\boldsymbol{\vartheta}^0) \le \left(\frac{1}{2N}\|\boldsymbol{A}\|_2\|\varphi(S;\boldsymbol{\alpha}^0)\|_2+\frac{\sqrt{\pi}}{N}\right)^2$$
$$\le \frac{1}{2}\left(1+3\gamma\sqrt{m}n^r\|\boldsymbol{A}\|_2\left(2\log\frac{4m(n+1)}{\delta}\right)^{(r+1)/2}\left(r\sqrt{2\log(2n)}+\sqrt{\log(8/\delta)/2}\right)\right)^2,$$

where the first inequality comes from the fact that $|f| \le 1$ by our assumption of the target function. □

The following lemma shows the positive definiteness of $\boldsymbol{G}^{(a)}$ at initialization.

Lemma B.5. *For any $\delta \in (0,1)$, if $m \ge \frac{16N^4C_n}{\lambda_S^2\delta}$, then with probability at least $1-\delta$ over the random initialization in* (30)*, we have*

$$\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)\right) \ge \frac{3}{4}\lambda_S,$$

where $C_n := \mathbf{E}\|\boldsymbol{w}\|_1^{4r} < +\infty$ with $\boldsymbol{w} \sim N(\boldsymbol{0}, \boldsymbol{I}_n)$.

Proof. We define $\Omega_{ij} := \{\boldsymbol{\vartheta}^0 \mid |\boldsymbol{G}_{ij}^{(a)}(\boldsymbol{\vartheta}^0) - \boldsymbol{K}_{ij}^{(a)}| \le \frac{\lambda_S}{4N}\}$. Note that

$$|g^{(a)}(\boldsymbol{w}_k^0;\boldsymbol{x}_i,\boldsymbol{x}_j)| \le \|\boldsymbol{w}_k^0\|_1^{2r}.$$

So

$$\mathrm{Var}\left[g^{(a)}(\boldsymbol{w}_k^0;\boldsymbol{x}_i,\boldsymbol{x}_j)\right] \le \mathbf{E}\left[g^{(a)}(\boldsymbol{w}_k^0;\boldsymbol{x}_i,\boldsymbol{x}_j)\right]^2 \le \mathbf{E}\|\boldsymbol{w}_k^0\|_1^{4r} = C_n,$$

and

$$\mathrm{Var}\left[\boldsymbol{G}_{ij}^{(a)}(\boldsymbol{\vartheta}^0)\right] = \frac{1}{m^2}\sum_{k=1}^m \mathrm{Var}\left[g^{(a)}(\boldsymbol{w}_k^0;\boldsymbol{x}_i,\boldsymbol{x}_j)\right] \le \frac{C_n}{m}.$$

Then the probability of the event Ω_{ij} has the lower bound:

$$\mathbf{P}(\Omega_{ij}) \ge 1 - \frac{\mathrm{Var}\left[\boldsymbol{G}_{ij}^{(a)}(\boldsymbol{\vartheta}^0)\right]}{[\lambda_S/(4N)]^2} \ge 1 - \frac{16N^2C_n}{\lambda_S^2 m}.$$

Thus, with probability at least $\left(1-\frac{16N^2C_n}{\lambda_S^2 m}\right)^{N^2} \ge 1 - \frac{16N^4C_n}{\lambda_S^2 m}$, we have all events Ω_{ij} for $i, j \in [N]$ to occur. This implies that with probability at least $1 - \frac{16N^4C_n}{\lambda_S^2 m}$, we have

$$\|\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0) - \boldsymbol{K}^{(a)}\|_F \le \frac{\lambda_S}{4}.$$

Note that $\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)$ and $\boldsymbol{K}^{(a)}$ are positive semi-definite normal matrices. Let $\boldsymbol{v}$ be the singular vector of $\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)$ corresponding to the smallest singular value, then

$$\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)\right) = \boldsymbol{v}^\top\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)\boldsymbol{v} = \boldsymbol{v}^\top\boldsymbol{K}^{(a)}\boldsymbol{v} + \boldsymbol{v}^\top(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0) - \boldsymbol{K}^{(a)})\boldsymbol{v} \ge \lambda_S - \|\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0) - \boldsymbol{K}^{(a)}\|_2 \ge \lambda_S - \|\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0) - \boldsymbol{K}^{(a)}\|_F.$$

So

$$\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)\right) \ge \lambda_S - \|\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0) - \boldsymbol{K}^{(a)}\|_F \ge \frac{3}{4}\lambda_S.$$

For any $\delta \in (0,1)$, if $m \ge \frac{16N^4C_n}{\lambda_S^2\delta}$, then with probability at least $1 - \frac{16N^4C_n}{\lambda_S^2 m} \ge 1 - \delta$ over the initialization $\boldsymbol{\vartheta}^0$, we have $\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)\right) \ge \frac{3}{4}\lambda_S$. □

The following lemma estimates the empirical loss dynamics before the stopping time $t^\square$ in (34).

Lemma B.6. *For any $\delta \in (0,1)$, if $m \geq \frac{16N^4 C_n}{\lambda_S^2 \delta}$, then with probability at least $1-\delta$ over the random initialization in (30), we have for any $t \in [0, t^\square)$*

$$R_S(\boldsymbol{\vartheta}(t)) \leq \exp\left(-\frac{m\lambda_S \lambda_{\boldsymbol{A}} t}{N}\right) R_S(\boldsymbol{\vartheta}^0).$$

Proof. From Lemma B.5, for any $\delta \in (0,1)$ with probability at least $1-\delta$ over initialization $\boldsymbol{\vartheta}^0$ and for any $t \in [0, t^\square)$ with $t^\square$ defined in (34), we have $\boldsymbol{\vartheta}(t) \in \mathrm{M}(\boldsymbol{\vartheta}^0)$ defined right after (34) and

$$\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta})\right) \geq \lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)\right) - \|\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}) - \boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)\|_F \geq \frac{3}{4}\lambda_S - \frac{1}{4}\lambda_S = \frac{1}{2}\lambda_S.$$

Note that $\boldsymbol{G}_{ij} = \frac{1}{m}\nabla_{\boldsymbol{\vartheta}}\varphi(\boldsymbol{x}_i;\boldsymbol{\vartheta}) \bullet \nabla_{\boldsymbol{\vartheta}}\varphi(\boldsymbol{x}_j;\boldsymbol{\vartheta})$ and $\nabla_{\boldsymbol{\vartheta}} R_S = \frac{1}{N}\nabla_{\boldsymbol{\vartheta}}\varphi(S;\boldsymbol{\vartheta})\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{e}$, so

$$\|\nabla_{\boldsymbol{\vartheta}} R_S(\boldsymbol{\vartheta}(t))\|_2^2 = \frac{m}{N^2}\boldsymbol{e}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{G}(\boldsymbol{\vartheta}(t))\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{e} \geq \frac{m}{N^2}\boldsymbol{e}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(t))\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{e},$$

where the last equation is true by the fact that $G^{(w)}(\boldsymbol{\vartheta}(t))$ is a Gram matrix and hence positive semi-definite. Together with

$$\begin{aligned}
\frac{m}{N^2}\boldsymbol{e}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(t))\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{e} &\geq \frac{m}{N^2}\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(t))\right)\boldsymbol{e}^\top \boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{A}^\top \boldsymbol{A}\boldsymbol{e}. \\
&\geq \frac{2m}{N}\lambda_{\min}\left(\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(t))\right)\lambda_{\min}\left(\boldsymbol{A}\boldsymbol{A}^\top\right) R_S(\boldsymbol{\vartheta}(t)) \\
&\geq \frac{m}{N}\lambda_S \lambda_{\boldsymbol{A}} R_S(\boldsymbol{\vartheta}(t)),
\end{aligned}$$

then finally we get

$$\frac{\mathrm{d}}{\mathrm{d}t} R_S(\boldsymbol{\vartheta}(t)) = -\|\nabla_{\boldsymbol{\vartheta}} R_S(\boldsymbol{\vartheta}(t))\|_2^2 \leq -\frac{m}{N}\lambda_S \lambda_{\boldsymbol{A}} R_S(\boldsymbol{\vartheta}(t)).$$

Integrating the above equation yields the conclusion in this lemma. □

The following lemma shows that the parameters in the two-layer neural network are uniformly bounded in time during the training before time $t^\square$ as defined in (34).

Lemma B.7. *For any $\delta \in (0,1)$, if*

$$m \geq \max\left\{\frac{32N^4 C_n}{\lambda_S^2 \delta}, \frac{5\max\{n,r\}N^2\left(2\lambda_{\boldsymbol{A},N} R_S(\boldsymbol{\vartheta}^0)\right)^{r-1}}{\lambda_S \lambda_{\boldsymbol{A}}} 2^n \left(2\log\frac{4m(n+1)}{\delta}\right)^r\right\},$$

then with probability at least $1-\delta$ over the random initialization in (30), for any $t \in [0, t^\square)$ and any $k \in [m]$,

$$\begin{aligned}
&|a_k(t) - a_k(0)| \leq q, \qquad \|\boldsymbol{w}_k(t) - \boldsymbol{w}_k(0)\|_\infty \leq q, \\
&|a_k(0)| \leq \gamma\eta, \qquad\qquad \|\boldsymbol{w}_k(0)\|_\infty \leq \eta,
\end{aligned}$$

where

$$q := 2n\left(2\log\frac{4m(n+1)}{\delta}\right)^r \frac{2\max\{n,r\}N^2\left(2\lambda_{\boldsymbol{A},N} R_S(\boldsymbol{\vartheta}^0)\right)^{\frac{r-1}{2}}}{m\lambda_S \lambda_{\boldsymbol{A}}}$$

and

$$\eta := \sqrt{2\log\frac{4m(n+1)}{\delta}}.$$

Proof. Let $\xi(t) = \max_{k\in[m], s\in[0,t]}\{|a_k(s)|, \|\boldsymbol{w}_k(s)\|_\infty\}$. Note that

$$\begin{aligned}
|\nabla_{a_k} R_S(\boldsymbol{\vartheta})|^2 &= \left|\frac{1}{N}\sum_{i=1}^N (\boldsymbol{e}^\top \boldsymbol{A}^\top \boldsymbol{A})_i \sigma(\boldsymbol{w}_k^\top \boldsymbol{x}_i)\right|^2 \\
&\leq \|\boldsymbol{w}_k\|_1^{2r}\left|\frac{1}{N}\sum_{i=1}^N (|\boldsymbol{e}^\top \boldsymbol{A}^\top \boldsymbol{A}|)_i\right|^2 \\
&\leq 2\|\boldsymbol{w}_k\|_1^{2r}\lambda_{\boldsymbol{A},N} R_S(\boldsymbol{\vartheta}) \\
&\leq 2n^{2r}(\xi(t))^{2r}\lambda_{\boldsymbol{A},N} R_S(\boldsymbol{\vartheta}),
\end{aligned}$$

where $\lambda_{\boldsymbol{A},N}$ denotes the largest eigenvalue of $\boldsymbol{A}$, and

$$\begin{aligned}
\|\nabla_{\boldsymbol{w}_k} R_S(\boldsymbol{\vartheta})\|_\infty^2 &= \left\| \frac{1}{N}\sum_{i=1}^N (e^{\boldsymbol{x}_i^\top \boldsymbol{A} \boldsymbol{x}_i})_i a_k \sigma'(\boldsymbol{w}_k^\top \boldsymbol{x}_i)\boldsymbol{x}_i \right\|_\infty^2 \\
&\le |a_k|^2 r^2 \|\boldsymbol{w}_k\|_1^{2(r-1)} \left\| \frac{1}{N}\sum_{i=1}^N (e^{\boldsymbol{x}_i^\top \boldsymbol{A} \boldsymbol{x}_i})_i \boldsymbol{x}_i \right\|_\infty^2 \\
&\le |a_k|^2 r^2 \|\boldsymbol{w}_k\|_1^{2(r-1)} \left| \frac{1}{N}\sum_{i=1}^N (|e^{\boldsymbol{x}_i^\top \boldsymbol{A} \boldsymbol{x}_i}|)_i \right|^2 \\
&\le 2|a_k|^2 r^2 \|\boldsymbol{w}_k\|_1^{2(r-1)} \lambda_{\boldsymbol{A},N} R_S(\boldsymbol{\vartheta}) \\
&\le 2r^2 n^{2(r-1)} (\xi(t))^{2r} \lambda_{\boldsymbol{A},N} R_S(\boldsymbol{\vartheta}).
\end{aligned}$$

From Lemma B.6, if $m \ge \frac{32N^4 C_n}{\lambda_S^2 \delta}$, then with probability at least $1-\delta/2$ over random initialization, one can represent (31) in an integral form and obtain,

$$\begin{aligned}
|a_k(t) - a_k(0)| &\le \int_0^t |\nabla_{a_k} R_S(\boldsymbol{\vartheta}(s))| \, ds \\
&\le \sqrt{2\lambda_{\boldsymbol{A},N}}\, n^r (\xi(t))^r \int_0^t \sqrt{R_S(\boldsymbol{\vartheta}(s))}\, ds \\
&\le \sqrt{2\lambda_{\boldsymbol{A},N}}\, n^r (\xi(t))^r \int_0^t \sqrt{R_S(\boldsymbol{\vartheta}^0)} \exp\left(-\frac{m\lambda_S \lambda_{\boldsymbol{A}} s}{2N}\right) ds \\
&\le \frac{2\sqrt{2\lambda_{\boldsymbol{A},N}}\, n^r N \sqrt{R_S(\boldsymbol{\vartheta}^0)}}{m\lambda_S \lambda_{\boldsymbol{A}}} (\xi(t))^r \\
&\le p(\xi(t))^r,
\end{aligned}$$

where $p := \frac{2\sqrt{2\lambda_{\boldsymbol{A},N}} \max\{n,r\} n^{r-1} N \sqrt{R_S(\boldsymbol{\vartheta}^0)}}{m\lambda_S \lambda_{\boldsymbol{A}}}$.

$$\begin{aligned}
\|\boldsymbol{w}_k(t) - \boldsymbol{w}_k(0)\|_\infty &\le \int_0^t \|\nabla_{\boldsymbol{w}_k} R_S(\boldsymbol{\vartheta}(s))\|_\infty \, ds \\
&\le \sqrt{2\lambda_{\boldsymbol{A},N}}\, r n^{r-1} (\xi(t))^r \int_0^t \sqrt{R_S(\boldsymbol{\vartheta}(s))}\, ds \\
&\le \sqrt{2\lambda_{\boldsymbol{A},N}}\, r n^{r-1} (\xi(t))^r \int_0^t \sqrt{R_S(\boldsymbol{\vartheta}^0)} \exp\left(-\frac{m\lambda_S \lambda_{\boldsymbol{A}} s}{2N}\right) ds \\
&\le \frac{2\sqrt{2\lambda_{\boldsymbol{A},N}}\, N r n^{r-1} \sqrt{R_S(\boldsymbol{\vartheta}^0)}}{m\lambda_S \lambda_{\boldsymbol{A}}} (\xi(t))^r \\
&\le p(\xi(t))^r.
\end{aligned}$$

We should point out that the above inequalities hold for all $t \in (0, t^*)$ since the upper bounds are based on Lemma B.6, thus

$$\xi(t) \le \xi(0) + p(\xi(t))^r, \tag{37}$$

for all $t \in (0, t^*)$. From Lemma B.3 with probability at least $1-\delta/2$,

$$\xi(0) = \max_{k\in[m]} \{|a_k(0)|, \|\boldsymbol{w}_k(0)\|_\infty\} \le \max\left(\sqrt{2\log\frac{4m(n+1)}{\delta}}, \sqrt{2\log\frac{4m(n+1)}{\delta}}\right) = \sqrt{2\log\frac{4m(n+1)}{\delta}} = \eta. \tag{38}$$

Since

$$m \ge \frac{5\max\{n,r\} N \sqrt{2\lambda_{\boldsymbol{A},N}} \sqrt{R_S(\boldsymbol{\vartheta}^0)}}{\lambda_S \lambda_{\boldsymbol{A}}} \left(2n\sqrt{2\log\frac{4m(n+1)}{\delta}}\right)^{r-1} = \frac{5}{2} mp(2\eta)^{r-1},$$

then $p(2\eta)^{r-1} \le \frac{2}{5}$. Let

$$t_0 := \inf\{t \mid \xi(t) > 2\eta\}.$$

Then t_0 is the first time for the magnitude of a NN parameter exceeding 2η. Recall that t^*, introduced in (34), denotes the first time for $\|\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}) - \boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}^0)\|_F > \frac{1}{4}\lambda_S$. We will prove $t_0 \ge t^*$, i.e., we show that, as long as the kernel

$\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(t))$ introduced by the gradient descent method is well controlled around its initialization $\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(0))$, all network parameters have a well-controlled magnitude in the sense that there is no parameter with a magnitude larger than 2η. We will prove $t_0 \geq t^*$ by contradiction. Suppose that $t_0 < t^*$. For $t \in [0, t_0)$, by (37), (38), and $\xi(t) \leq 2\eta$, we have

$$\xi(t) \leq \eta + p(2\eta)^{r+1}\xi(t) \leq \eta + \frac{2}{5}\xi(t),$$

then

$$\xi(t) \leq \frac{5}{3}\eta.$$

After letting $t \to t_0$, the inequality just above contradicts with the definition of t_0. So $t_0 \geq t^*$ and then $\xi(t) \leq 2\eta$ for all $t \in [0, t^*)$. Thus

$$\begin{aligned} |a_k(t) - a_k(0)| &\leq (2\eta)^r p \\ \|\boldsymbol{w}_k(t) - \boldsymbol{w}_k(0)\|_\infty &\leq (2\eta)^r p. \end{aligned}$$

Finally, notice that

$$(2\eta)^r p = \left(2n \sqrt{2\log\frac{4m(n+1)}{\delta}}\right)^r \frac{2\max\{n,r\}N\sqrt{2\lambda_{\boldsymbol{A},N}R_S(\boldsymbol{\vartheta}^0)}}{nm\lambda_S\lambda_{\boldsymbol{A}}} = q, \tag{39}$$

which ends the proof. □

B.4 Proof of Theorem 4.2

Now we are ready to prove Theorem 4.2.

Proof of Theorem 4.2. From Lemma B.6, it is sufficient to prove that the stopping time t^* in Lemma B.6 is equal to $+\infty$. We will prove this by contradiction.

Suppose $t^* < +\infty$. Note that

$$|\boldsymbol{G}_{ij}^{(a)}(\boldsymbol{\vartheta}(t^*)) - \boldsymbol{G}_{ij}^{(a)}(\boldsymbol{\vartheta}(0))| \leq \frac{1}{m}\sum_{k=1}^m |g^{(a)}(\boldsymbol{w}_k(t^*); \boldsymbol{x}_i, \boldsymbol{x}_j) - g^{(a)}(\boldsymbol{w}_k(0); \boldsymbol{x}_i, \boldsymbol{x}_j)|. \tag{40}$$

By the mean value theorem,

$$|g^{(a)}(\boldsymbol{w}_k(t^*); \boldsymbol{x}_i, \boldsymbol{x}_j) - g^{(a)}(\boldsymbol{w}_k(0); \boldsymbol{x}_i, \boldsymbol{x}_j)| \leq \left\|\nabla_{\boldsymbol{w}} g^{(a)}\left(c\boldsymbol{w}_k(t^*) + (1-c)\boldsymbol{w}_k(0); \boldsymbol{x}_i, \boldsymbol{x}_j\right)\right\|_\infty \|\boldsymbol{w}_k(t^*) - \boldsymbol{w}_k(0)\|_1$$

for some $c \in (0,1)$. Further computation yields

$$\nabla_{\boldsymbol{w}} g^{(a)}(\boldsymbol{w}; \boldsymbol{x}_i, \boldsymbol{x}_j) = \left[\sigma'(\boldsymbol{w}^\top \boldsymbol{x}_i)\boldsymbol{x}_i \times \left[\sigma(\boldsymbol{w}^\top \boldsymbol{x}_j) + \left[\sigma'(\boldsymbol{w}^\top \boldsymbol{x}_j)\boldsymbol{x}_j \times \left[\sigma(\boldsymbol{w}^\top \boldsymbol{x}_i)\right]\right]\right]\right]$$

for all $\boldsymbol{w}$. Hence, it holds for all $\boldsymbol{w}$ that $\|\nabla_{\boldsymbol{w}} g^{(a)}(\boldsymbol{w}; \boldsymbol{x}_i, \boldsymbol{x}_j)\|_\infty \leq 2r\|\boldsymbol{w}\|_1^{2r+1}$. Therefore, the bound in (40) becomes

$$|\boldsymbol{G}_{ij}^{(a)}(\boldsymbol{\vartheta}(t^*)) - \boldsymbol{G}_{ij}^{(a)}(\boldsymbol{\vartheta}(0))| \leq \frac{2r}{m}\sum_{k=1}^m \|c\boldsymbol{w}_k(t^*) + (1-c)\boldsymbol{w}_k(0)\|_1^{2r+1}\|\boldsymbol{w}_k(t^*) - \boldsymbol{w}_k(0)\|_1. \tag{41}$$

By Lemma B.7,

$$\|c\boldsymbol{w}_k(t^*) + (1-c)\boldsymbol{w}_k(0)\|_1 \leq \|\boldsymbol{w}_k(0)\|_1 + \|\boldsymbol{w}_k(t^*) - \boldsymbol{w}_k(0)\|_1 \leq n(\eta + q) \leq 2n\eta,$$

where η and q are defined in Lemma B.7. So, (41) and the above inequalities indicate

$$|\boldsymbol{G}_{ij}^{(a)}(\boldsymbol{\vartheta}(t^*)) - \boldsymbol{G}_{ij}^{(a)}(\boldsymbol{\vartheta}(0))| \leq r(2n)^{2r}\eta^{2r+1}q,$$

and

$$\begin{aligned} \|\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(t^*)) - \boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(0))\|_F &\leq nr(2n)^{2r}\eta^{2r+1}q \\ &= nr(2n)^{2r}\left(2\log\frac{4m(n+1)}{\delta}\right)^{(3r+1)/2}(2n)^r \frac{2\max\{n,r\}N\sqrt{2\lambda_{\boldsymbol{A},N}R_S(\boldsymbol{\vartheta}^0)}}{nm\lambda_S\lambda_{\boldsymbol{A}}} \\ &= 2^{9r/2+1}n^{3r+1}N^2r^4\left(\log\frac{4m(n+1)}{\delta}\right)^{(3r+1)/2}\frac{\max\{n,r\}\sqrt{\lambda_{\boldsymbol{A},N}R_S(\boldsymbol{\vartheta}^0)}}{m\lambda_S\lambda_{\boldsymbol{A}}} \\ &\leq \frac{1}{4}\lambda_S, \end{aligned}$$

if we choose

$$m \geq 2^{9r/2+3} n^{3r+1} N^2 r \left(\log \frac{4m(n+1)}{\delta} \right)^{(3r+1)/2} \frac{\max\{n,r\} \sqrt{\lambda_{\boldsymbol{A},N} R_S(\boldsymbol{\vartheta}^0)}}{\lambda_S^2 \lambda_{\boldsymbol{A}}}.$$

The fact that $\|\boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(t^*)) - \boldsymbol{G}^{(a)}(\boldsymbol{\vartheta}(0))\|_F \leq \frac{1}{4}\lambda_S$ above contradicts with the definition of t^* in (34).

Let us summarize the conclusion in the above discussion. Let $C_n := \mathbf{E}\|\boldsymbol{w}\|_1^{4r} < +\infty$ with $\boldsymbol{w} \sim N(\boldsymbol{0}, \boldsymbol{I}_n)$. The largest eigenvalue and the condition number of $\boldsymbol{A}\boldsymbol{A}^T$ are denoted as $\lambda_{\boldsymbol{A},N}$ and κ_A, respectively. For any $\delta \in (0,1)$, define

$$m_1 = \frac{32 N^4 C_n}{\lambda_S^2 \delta},$$

$$m_2 = \frac{5\max\{n,r\} N \sqrt{2\lambda_{\boldsymbol{A},N} R_S(\boldsymbol{\vartheta}^0)}}{\lambda_S \lambda_{\boldsymbol{A}}} \left(2n \sqrt{2 \log \frac{4m(n+1)}{\delta}} \right)^{r+1},$$

and

$$m_3 = 2^{9r/2+3} n^{3r+1} N^2 r \left(\log \frac{4m(n+1)}{\delta} \right)^{(3r+1)/2} \frac{\max\{n,r\} \sqrt{\lambda_{\boldsymbol{A},N} R_S(\boldsymbol{\vartheta}^0)}}{\lambda_S^2 \lambda_{\boldsymbol{A}}}. \tag{42}$$

Then when $m \geq \max\{m_1, m_2, m_3\}$, with probability at least $1-\delta$ over the random initialization $\boldsymbol{\vartheta}^0$, we have, for all $t \geq 0$,

$$R_S(\boldsymbol{\vartheta}(t)) \leq \exp\left(-\frac{m\lambda_S \lambda_{\boldsymbol{A}} t}{N} \right) R_S(\boldsymbol{\vartheta}^0).$$

Note that $O(\kappa_A \mathrm{poly}(N, r, n, \frac{1}{\delta}, \frac{1}{\lambda_S})) \geq \max\{m_1, m_2, m_3\}$. Hence, we have completed the proof. □

C More details on the numerical experiments

In this Appendix, we report the detailed hyper-parameter setting for the six examples presented in the main text. For convenience, we also report the notations in Table 5.

Notation	Explanation
k	k-nearest neighbors in DM
k_2	variable bandwidth parameter in VBDM
ϵ	the bandwidth parameter for local integral in DM
m	the width of hidden layer in FNN
T	the number of iterations for training a FNN
N	the number of training points
N_b	the number of training points on the boundary
γ	the regularization coefficient for $\frac{1}{2}\|\boldsymbol{\varphi}_{\boldsymbol{\vartheta}}\|_2^2$ introduced in Section 3
λ	the penalty coefficient to enforce the boundary condition in the loss function (9)

Table 5: The summary of hyperparameter notations in the algorithms.

	N	625	1,225	2,500	5,041	10,000
VBDM	ϵ	0.0146	0.0078	0.0068	0.0036	0.0026
	k	100	100	200	200	300
	k_2	30	30	60	600	90
NN	T	2k	3k	4k	6k	8k
	m	50γ	71	100	141	200
	0.001		0.001	0.001	0.001	0.001

Table 6: The hyperparameter setting for **Example 1**: 2D Sphere.

	N	625	1225	2500	5041	10000	19881	40000	80089
DM	ϵ	0.1166	0.0508	0.0237	0.0118	0.0059	0.0029	0.0014	0.0008
	k	128	128	128	256	256	256	512	768
NN	T	2000	3000	4000	4000	4000	4000	4000	8000
	m	50	71	100	141	200	282	400	583
	γ	0.001	0.001	0.001	0.001	0.001	0.002	0.005	0.01

Table 7: The hyperparameter setting for **Example 2**: 2D torus embedded in $\mathbf{R}^3$.

	N	1024	2025	4096	16384	32400
DM	ϵ	0.0221	0.0096	0.0048	0.0013	0.00064
	k	128	128	128	256	256
NN	T	2000	3000	4000	10000	12000
	m	100	100	150	250	250
	λ	5.0	5.0	5.0	5.0	5.0

Table 8: The hyperparameter setting for **Example 3**: 2D semi-torus with the Dirichlet condition.

	N	512	1331	4096	12167	24389
DM	ϵ	0.43	0.23	0.12	0.073	0.051
	k	128	128	256	256	256
NN	T	1000	2000	2000	3000	3000
	m	100	150	250	400	500
	γ	0.001	0.001	0.005	0.005	0.005

Table 9: The hyperparameter setting for **Example 4**: 3D manifold embedded in $\mathbf{R}^{12}$.

	N	4326	8650	17299	34594
VBDM	ϵ	0.0048	0.0024	0.0012	0.0006
	k	100	100	100	100
	k_2	30	30	30	30
NN	T	30k	40k	50k	100k
	m	100	150	150	150

Table 10: The hyperparameter setting for **Example 5**: Bunny.

	N	2185	4340	8261	17157
VBDM	ϵ	0.000793	0.000370	0.000185	0.000081
	k	60	60	60	60
	k_2	20	20	20	20
NN	T	100k	100k	100k	200k
	m	100	100	200	500
	Activation	ReLU	ReLU	ReLU	Tanh
	Layer	4	4	6	6
	λ	1	1	10	10000

Table 11: The hyperparameter setting for **Example 6**: Face with the Dirichlet condition.

References

[1] The stanford 3d scanning repository. http://graphics.stanford.edu/data/3Dscanrep/.

[2] Romeo Alexander and Dimitrios Giannakis. Operator-theoretic framework for forecasting nonlinear time series with kernel analog techniques. *Physica D: Nonlinear Phenomena*, 409:132520, 2020.

[3] Zeyuan Allen-Zhu, Yuanzhi Li, and Zhao Song. A convergence theory for deep learning via over-parameterization. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 242–252. PMLR, 09–15 Jun 2019.

[4] Sanjeev Arora, Simon S. Du, Wei Hu, Zhiyuan Li, and Ruosong Wang. Fine-grained analysis of optimization and generalization for overparameterized two-layer neural networks. In *36th International Conference on Machine Learning, ICML 2019*, 36th International Conference on Machine Learning, ICML 2019, pages 477–502. International Machine Learning Society (IMLS), January 2019. 36th International Conference on Machine Learning, ICML 2019 ; Conference date: 09-06-2019 Through 15-06-2019.

[5] A. R. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Transactions on Information Theory*, 39(3):930–945, 1993.

[6] Christian Beck, Sebastian Becker, Patrick Cheridito, Arnulf Jentzen, and Ariel Neufeld. Deep splitting method for parabolic PDEs. *arXiv e-prints*, arXiv:1907.03452, Jul 2019.

[7] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28 – 41, 2018.

[8] Julius Berner, Philipp Grohs, and Arnulf Jentzen. Analysis of the generalization error: Empirical risk minimization over deep artificial neural networks overcomes the curse of dimensionality in the numerical approximation of black–scholes partial differential equations. *SIAM Journal on Mathematics of Data Science*, 2(3):631–657, 2020.

[9] T. Berry and J. Harlim. Variable bandwidth diffusion kernels. *Appl. Comput. Harmon. Anal.*, 40:68–96, 2016.

[10] Tyrus Berry and Timothy Sauer. Consistent manifold representation for topological data analysis. *Found. Data Sci.*, 1(1):1, 2019.

[11] Marcelo Bertalmıo, Li-Tien Cheng, Stanley Osher, and Guillermo Sapiro. Variational problems and partial differential equations on implicit surfaces. *Journal of Computational Physics*, 174(2):759–780, 2001.

[12] Andrea Bonito, J Manuel Cascón, Khamron Mekchay, Pedro Morin, and Ricardo H Nochetto. High-order afem for the laplace–beltrami operator: Convergence rates. *Foundations of Computational Mathematics*, 16(6):1473–1539, 2016.

[13] Fernando Camacho and Alan Demlow. L2 and pointwise a posteriori error estimates for fem for elliptic pdes on surfaces. *IMA Journal of Numerical Analysis*, 35(3):1199–1227, 2015.

[14] Jay Chu and Richard Tsai. Volumetric variational principles for a class of partial differential equations defined on surfaces and curves. *Research in the Mathematical Sciences*, 5(2):1–38, 2018.

[15] A. Cichocki and R. Unbehauen. Neural networks for solving systems of linear equations and related problems. *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, 39(2):124–138, 1992.

[16] R. Coifman and S. Lafon. Diffusion maps. *Appl. Comput. Harmon. Anal.*, 21:5–30, 2006.

[17] Ronald R Coifman, Yoel Shkolnisky, Fred J Sigworth, and Amit Singer. Graph Laplacian tomography from unknown random projections. *Image Processing, IEEE Transactions on*, 17(10):1891–1899, 2008.

[18] Keenan Crane. Keenan's 3d model repository. http://www.cs.cmu.edu/ kmcrane/Projects/ModelRepository/.

[19] I. Daubechies, R. DeVore, S. Foucart, B. Hanin, and G. Petrova. Nonlinear approximation and (deep) relu networks. *arxiv:1905.02199*, , 2019.

[20] Ronald DeVore, Boris Hanin, and Guergana Petrova. Neural network approximation. *Acta Numerica*, 30:327–444, 2021.

[21] Qiang Du, Yiqi Gu, Haizhao Yang, and Chao Zhou. The discovery of dynamics via linear multistep methods and deep learning: error estimation. *SIAM Journal on Numerical Analysis*, 60(4):2014–2045, 2022.

[22] Simon Du, Jason Lee, Haochuan Li, Liwei Wang, and Xiyu Zhai. Gradient descent finds global minima of deep neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 1675–1685. PMLR, 09–15 Jun 2019.

[23] Simon S. Du, Xiyu Zhai, Barnabas Poczos, and Aarti Singh. Gradient descent provably optimizes over-parameterized neural networks. In *International Conference on Learning Representations*, 2019.

[24] Chenguang Duan, Yuling Jiao, Yanming Lai, Dingwei Li, Xiliang Lu, and Jerry Zhijian Yang. Convergence Rate Analysis for Deep Ritz Method. *Communications in Computational Physics*, 31(4):1020–1048, 2022.

[25] David B Dunson, Hau-Tieng Wu, and Nan Wu. Spectral convergence of graph Laplacian and heat kernel reconstruction in L^∞ from random samples. *Applied and Computational Harmonic Analysis*, 55:282–336, 2021.

[26] Gerhard Dziuk and Charles M Elliott. Finite element methods for surface pdes. *Acta Numerica*, 22:289–396, 2013.

[27] Weinan E, Jiequn Han, and Arnulf Jentzen. Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Communications in Mathematics and Statistics*, 5(4):349–380, Dec 2017.

[28] Weinan E, Chao Ma, and Lei Wu. A priori estimates of the population risk for two-layer neural networks. *Communications in Mathematical Sciences*, 17(5):1407 – 1425, 2019.

[29] Weinan E, Chao Ma, and Lei Wu. Barron Spaces and the Compositional Function Spaces for Neural Network Models. *Constructive Approximation*, , 2020.

[30] Weinan E and Qingcan Wang. Exponential convergence of the deep neural network approximation for analytic functions. *Science China Mathematics*, 61(10):1733–1740, 10 2018.

[31] Charles M Elliott and Björn Stinner. Modeling and computation of two phase geometric biomembranes using surface finite elements. *Journal of Computational Physics*, 229(18):6585–6612, 2010.

[32] Björn Engquist and Lexing Ying. A fast directional algorithm for high frequency acoustic scattering in two dimensions. *Communications in Mathematical Sciences*, 7:327–345, 2009.

[33] Z. Fang and J. Zhan. A physics-informed neural network framework for pdes on 3d surfaces: Time independent problems. *IEEE Access*, 8:26328–26335, 2020.

[34] Edward J Fuselier and Grady B Wright. A high-order kernel method for diffusion and reaction-diffusion equations on surfaces. *Journal of Scientific Computing*, 56(3):535–565, 2013.

[35] Michael Garland and Paul S Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 209–216, 1997.

[36] F. Gilani and J. Harlim. Approximating solutions of linear elliptic PDE's on smooth manifold using local kernels. *J. Comput. Phys.*, 395:563–582, 2019.

[37] David Gilbarg and Neil S Trudinger. *Elliptic partial differential equations of second order*. springer, 2015.

[38] Yiqi Gu and Michael K. Ng. Deep neural networks for solving large linear systems arising from high-dimensional problems. *SIAM Journal on Scientific Computing*, 45(5):A2356–A2381, 2023.

[39] Ingo Gühring, Gitta Kutyniok, and Philipp Petersen. Error bounds for approximations with deep relu neural networks in ws,p norms. *Analysis and Applications*, 18(05):803–859, 2020.

[40] N. Halko, P. G. Martinsson, and J. A. Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM Review*, 53(2):217–288, 2011.

[41] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[42] Jiequn Han and Jihao Long. Convergence of the deep bsde method for coupled fbsdes. *Probability, Uncertainty and Quantitative Risk*, 5(1):5, 2020.

[43] Qing Han and Fanghua Lin. *Elliptic partial differential equations*, volume 1. American Mathematical Soc., 2011.

[44] J. Harlim. *Data-Driven Computational Methods: Parameter and Operator Estimations*. Cambridge University Press, 2018.

[45] John Harlim, Daniel Sanz-Alonso, and Ruiyi Yang. Kernel methods for bayesian elliptic inverse problems on manifolds. *SIAM/ASA Journal on Uncertainty Quantification*, 8(4):1414–1445, 2020.

[46] Matthias Hein, Jean-Yves Audibert, and Ulrike von Luxburg. Graph Laplacians and their convergence on random neighborhood graphs. *Journal of Machine Learning Research*, 8(6), 2007.

[47] Qingguo Hong, Jonathan W. Siegel, and Jinchao Xu. A Priori Analysis of Stable Neural Network Solutions to Numerical PDEs. *arxiv:2104.02903*, , 2021.

[48] Martin Hutzenthaler, Arnulf Jentzen, Thomas Kruse, and Tuan Anh Nguyen. A proof that rectified deep neural networks overcome the curse of dimensionality in the numerical approximation of semilinear heat equations. *SN Partial Differential Equations and Applications*, 1(10), 2020.

[49] Martin Hutzenthaler, Arnulf Jentzen, and von Wurstemberger Wurstemberger. Overcoming the curse of dimensionality in the approximative pricing of financial derivatives with default risks. *Electron. J. Probab.*, 25:73 pp., 2020.

[50] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural tangent kernel: Convergence and generalization in neural networks. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 8580–8589, Red Hook, NY, USA, 2018. Curran Associates Inc.

[51] Shixiao Willing Jiang and John Harlim. Ghost point diffusion maps for solving elliptic pdes on manifolds with classical boundary conditions. *Communications on Pure and Applied Mathematics*, 76(2):337–405, 2023.

[52] Yuehaw Khoo, Jianfeng Lu, and Lexing Ying. Solving parametric pde problems with artificial neural networks. *European Journal of Applied Mathematics*, :1–15, 2020.

[53] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In Yoshua Bengio and Yann LeCun, editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015.

[54] John M Lee. *Introduction to Smooth Manifolds*. Springer, 2013.

[55] Randall J LeVeque. *Finite difference methods for ordinary and partial differential equations: steady-state and time-dependent problems*, volume 98. Siam, 2007.

[56] Ke Li, Kejun Tang, Tianfan Wu, and Qifeng Liao. D3M: A Deep Domain Decomposition Method for Partial Differential Equations. *IEEE Access*, 8:5283–5294, 2020.

[57] Zhen Li and Zuoqiang Shi. A convergent point integral method for isotropic elliptic equations on a point cloud. *Multiscale Modeling & Simulation*, 14(2):874–905, 2016.

[58] Jian Liang, Rongjie Lai, Tsz Wai Wong, and Hongkai Zhao. Geometric understanding of point clouds using laplace-beltrami operator. In *2012 IEEE conference on computer vision and pattern recognition*, pages 214–221. IEEE, 2012.

[59] Senwei Liang, Liyao Lyu, Chunmei Wang, and Haizhao Yang. Reproducing activation function for deep learning. *arxiv:2101.04844*, , 2021.

[60] William E Lorensen and Harvey E Cline. Marching cubes: A high resolution 3d surface construction algorithm. *ACM siggraph computer graphics*, 21(4):163–169, 1987.

[61] Jianfeng Lu and Yulong Lu. A priori generalization error analysis of two-layer neural networks for solving high dimensional schrödinger eigenvalue problems. *Communications of the American Mathematical Society*, 2(1):1–21, 2022.

[62] Jianfeng Lu, Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network approximation for smooth functions. *SIAM Journal on Mathematical Analysis*, 53(5):5465–5506, 2021.

[63] Yiping Lu, Chao Ma, Yulong Lu, Jianfeng Lu, and Lexing Ying. A mean field analysis of deep ResNet and beyond: Towards provably optimization via overparameterization from depth. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 6426–6436. PMLR, 13–18 Jul 2020.

[64] Yulong Lu, Jianfeng Lu, and Min Wang. A priori generalization analysis of the deep ritz method for solving high dimensional elliptic partial differential equations. In Mikhail Belkin and Samory Kpotufe, editors, *Proceedings of Thirty Fourth Conference on Learning Theory*, volume 134 of *Proceedings of Machine Learning Research*, pages 3196–3241. PMLR, 15–19 Aug 2021.

[65] Tao Luo and Haizhao Yang. Two-layer neural networks for partial differential equations: Optimization and generalization theory. *ArXiv*, abs/2006.15733, 2020.

[66] Ilay Luz, Meirav Galun, Haggai Maron, Ronen Basri, and Irad Yavneh. Learning algebraic multigrid using graph neural networks. In *International Conference on Machine Learning*, pages 6489–6499. PMLR, 2020.

[67] Colin B Macdonald and Steven J Ruuth. The implicit closest point method for the numerical solution of partial differential equations on surfaces. *SIAM Journal on Scientific Computing*, 31(6):4330–4350, 2010.

[68] Lindsay Martin and Yen-Hsi Richard Tsai. Equivalent extensions of hamilton–jacobi–bellman equations on hypersurfaces. *Journal of Scientific Computing*, 84(3):1–29, 2020.

[69] Song Mei, Andrea Montanari, and Phan-Minh Nguyen. A mean field view of the landscape of two-layer neural networks. *Proceedings of the National Academy of Sciences*, 115(33):E7665–E7671, 2018.

[70] Facundo Mémoli, Guillermo Sapiro, and Paul Thompson. Implicit brain imaging. *NeuroImage*, 23:S179–S188, 2004.

[71] Meshlab. Meshlab. https://www.meshlab.net/.

[72] Hadrien Montanelli and Qiang Du. New error bounds for deep relu networks using sparse grids. *SIAM Journal on Mathematics of Data Science*, 1(1), Jan 2019.

[73] Hadrien Montanelli and Haizhao Yang. Error bounds for deep ReLU networks using the Kolmogorov–Arnold superposition theorem. *Neural Networks*, 129:1–6, 2020.

[74] Haizhao Montanelli, HadrienYang and Qiang Du. Deep relu networks overcome the curse of dimensionality for generalized bandlimited functions. *Journal of Computational Mathematics*, 39(6):801–815, 2021.

[75] Evert J Nyström. Über die praktische auflösung von integralgleichungen mit anwendungen auf randwertaufgaben. *Acta Mathematica*, 54(1):185–204, 1930.

[76] J Wilson Peoples and John Harlim. Spectral convergence of symmetrized graph laplacian on manifolds with boundary. *arXiv preprint arXiv:2110.06988*, 2021.

[77] Philipp Petersen and Felix Voigtlaender. Optimal approximation of piecewise smooth functions using deep ReLU neural networks. *Neural Networks*, 108:296 – 330, 2018.

[78] Cécile Piret. The orthogonal gradients method: A radial basis functions method for solving partial differential equations on arbitrary surfaces. *Journal of Computational Physics*, 231(14):4662–4675, 2012.

[79] 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.

[80] Matthias Rauter and Željko Tuković. A finite area scheme for shallow granular flows on three-dimensional surfaces. *Computers & Fluids*, 166:184–199, 2018.

[81] Steven J Ruuth and Barry Merriman. A simple embedding method for solving partial differential equations on surfaces. *Journal of Computational Physics*, 227(3):1943–1961, 2008.

[82] S. Shalev-Shwartz and S. Ben-David. *Understanding machine learning: From theory to algorithms*. Cambridge university press, 2014.

[83] V. Shankar, G. B. Wright, R. M. Kirby, and A. L. Fogelson. A radial basis function (rbf)-finite difference (fd) method for diffusion and reaction-diffusion equations on surfaces. *Journal of Scientific Computing*, 63(3), 2014.

[84] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network approximation characterized by number of neurons. *Communications in Computational Physics*, 28(5):1768–1811, 2020.

[85] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Deep network with approximation error being reciprocal of width to power of square root of depth. *Neural Computation*, 33(4):1005–1036, 2021.

[86] Zuowei Shen, Haizhao Yang, and Shijun Zhang. Neural network approximation: Three hidden layers are enough. *Neural Networks*, 141:160–173, 2021.

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

[88] Yeonjong Shin, Jerome Darbon, and George Em Karniadakis. On the convergence of physics informed neural networks for linear second-order elliptic and parabolic type pdes. *Communications in Computational Physics*, 28(5):2042–2074, 2020.

[89] Jonathan W. Siegel and Jinchao Xu. Approximation rates for neural networks with general activation functions. *Neural Networks*, 128:313 – 321, 2020.

[90] Amit Singer. From graph to manifold Laplacian: The convergence rate. *Appl. Comp. Harmonic Anal.*, 21:128–134, 2006.

[91] Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339 – 1364, 2018.

[92] Epifanio G Virga. *Variational theories for liquid crystals*. CRC Press, 2018.

[93] Shawn W Walker. Felicity: A matlab/c++ toolbox for developing finite element methods and simulation modeling. *SIAM Journal on Scientific Computing*, 40(2):C234–C257, 2018.

[94] Qile Yan, Shixiao Jiang, and John Harlim. Kernel-based methods for solving time-dependent advection-diffusion equations on manifolds. *J. Sci. Comput.*, 94(1), 2023.

[95] Haizhao Yang and Lexing Ying. A fast algorithm for multilinear operators. *Applied and Computational Harmonic Analysis*, 33(1):148–158, 2012.

[96] Yunfei Yang, Zhen Li, and Yang Wang. Approximation in shift-invariant spaces with deep relu neural networks. *Neural Networks*, 153:269–281, 2022.

[97] Dmitry Yarotsky. Error bounds for approximations with deep ReLU networks. *Neural Networks*, 94:103 – 114, 2017.

[98] Dmitry Yarotsky. Optimal approximation of continuous functions by very deep ReLU networks. In Sébastien Bubeck, Vianney Perchet, and Philippe Rigollet, editors, *Proceedings of the 31st Conference On Learning Theory*, volume 75 of *Proceedings of Machine Learning Research*, pages 639–649. PMLR, 06–09 Jul 2018.

[99] Dmitry Yarotsky and Anton Zhevnerchuk. The phase diagram of approximation rates for deep neural networks. In H. Larochelle, M. Ranzato, R. Hadsell, M. F. Balcan, and H. Lin, editors, *Advances in Neural Information Processing Systems*, volume 33, pages 13005–13015. Curran Associates, Inc., 2020.

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

[101] Lihi Zelnik-Manor and Pietro Perona. Self-tuning spectral clustering. *Advances in neural information processing systems*, 17, 2004.