A MATHEMATICAL PERSPECTIVE OF MACHINE LEARNING

WEINAN E

ABSTRACT

What lies at the heart of modern neural network-based machine learning is the ability to approximate very high dimensional functions with good accuracy. This opens up two major avenues of research. The first is to develop machine learning-based algorithms for scientific problems that suffer from the *curse of dimensionality*. The second is to build a theoretical framework that helps us to form a better foundation for machine learning. For the latter, the most important questions that need to be addressed include: Why do neural network models work so well in high dimension? Why does their performance depend so sensitively on the choice of the hyperparameters? Can we develop more robust and equally accurate new machine learning models and algorithms? In this article, we review some of the major progresses made in these directions.

MATHEMATICS SUBJECT CLASSIFICATION 2020

Primary 65; Secondary 68, 41, 60, 91, 93

KEYWORDS

Machine learning, scientific computing, curse of dimensionality, approximation theory, neural networks, error analysis, continuous formulation, integral differential equations

ICM INTERNATIONAL CONGRESS OF MATHEMATICIANS 2022 JULY 6—14

Proc. Int. Cong. Math. 2022, Vol. 2, pp. 914–954
DOI 10.4171/ICM2022/155

1. INTRODUCTION

Supervised learning. We begin with the simplest task in machine learning (ML), supervised learning. The goal is to approximate an unknown target function from a finite training dataset. Denote by $f^* : X = [0,1]^d \to \mathbb{R}$ the target function. Let $S = \{(\boldsymbol{x}_j, y_j = f^*(\boldsymbol{x}_j)), j \in [n] = \{1, 2, \ldots, n\}\}$ be the available dataset. Our objective is to approximate f^* as accurately as we can. This usually means that we would like to minimize the *population risk* in a given function class:

$$\mathcal{R}(f) = \mathbb{E}\big(f(\boldsymbol{x}) - f^*(\boldsymbol{x})\big)^2 = \int_X \big(f(\boldsymbol{x}) - f^*(\boldsymbol{x})\big)^2 d\mu,$$

where μ is a given probability distribution on X.

A typical supervised learning algorithm consists of the following three major components:

- Defining a *hypothesis space*. This is a set of functions that we use to approximate f^*. It is the analog of the finite element trial function space, except that in modern ML, we typically use neural network functions as the trial functions. We will use $\mathcal{H}_m$ to denote the hypothesis space where m is roughly the dimension of $\mathcal{H}_m$. We denote the functions in $\mathcal{H}_m$ generically as $f(\cdot, \theta)$ and we use θ to parametrize the functions in $\mathcal{H}_m$.

- Setting up an optimization problem for finding the optimal parameters. Though we are interested in minimizing the population risk, in practice we have to work with the *empirical risk* (or its variants):

  $$\mathcal{R}_n(\theta) = \frac{1}{n}\sum_j \big(f(\boldsymbol{x}_j, \theta) - y_j\big)^2 = \frac{1}{n}\sum_j \big(f(\boldsymbol{x}_j, \theta) - f^*(\boldsymbol{x}_j)\big)^2$$

  or, more generally,

  $$\mathcal{R}_n(\theta) = \frac{1}{n}\sum_j \ell_j(\theta),$$

  where the term ℓ_j is the loss for the jth data point. Regularization terms are sometimes added to this expression. The population and empirical risks are more commonly referred to as the training and testing errors, respectively.
  The difference between the true objective, the population risk, and the objective function that we work with in practice, the empirical risk, is an important issue that differentiates the optimization problems in ML from those in other settings.

- Solving this optimization problem. The simplest idea is to use the gradient descent algorithm (GD),

  $$\theta_{k+1} = \theta_k - \eta \nabla \mathcal{R}_n(\theta_k) = \theta_k - \eta \frac{1}{n}\sum_j \nabla \ell_j(\theta_k),$$

  where η is called the learning rate. Since the full gradient is an average over all training samples and is costly to evaluate, in practice, one often randomly selects

one term in that average and uses it instead of the full gradient. This leads to the stochastic gradient descent algorithm (SGD),

$$\theta_{k+1} = \theta_k - \eta \nabla \ell_{j_k}(\theta_k),$$

where $j_1, j_2, \ldots$ are i.i.d. random variables uniformly drawn from $\{1, 2, \ldots, n\}$. One of the main mysteries in ML is that SGD is not only more efficient than GD, it often leads to a smaller test error.

How do we choose the hypothesis space? In classical numerical algorithms, we choose polynomials, piecewise polynomials, wavelets, and the like. In linear regression, we choose functions of the form $f(\boldsymbol{x}) = \beta \cdot \boldsymbol{x} + \beta_0$, where β and β_0 are the parameters to be found. Neural network models are the most popular choice in modern ML. A simple neural network model takes the form $f(\boldsymbol{x}) = \sum_k a_k \sigma(\boldsymbol{w}_k \cdot \boldsymbol{x} + c_k)$, where σ is some scalar nonlinear function, called the activation function. Popular choices of σ include $\sigma(x) = \max(x, 0)$, the ReLU (rectified linear units) function, and $\sigma(x) = (1 + e^{-x})^{-1}$, the sigmoid function. This is called a two-layer neural network model since there are two affine transformations (represented by the parameters $\{a_k\}$ and $\{\boldsymbol{w}_k\}$, respectively) involved. As is usually the case in ML, we have neglected the constant terms in the affine transformations. To include them, one can think of $\boldsymbol{x}$ as being $(\boldsymbol{x}^T, 1)^T$ and change the dimensionality accordingly. We will adopt this convention throughout this report. Multilayer neural network models, or deep neural networks (DNN), are formed by compositions of functions of the form above:

$$f(\boldsymbol{x}, \theta) = \boldsymbol{W}_L \sigma \circ \big(\boldsymbol{W}_{L-1} \sigma \circ \big(\cdots \sigma \circ (\boldsymbol{W}_0 \boldsymbol{x})\big)\big), \quad \theta = (\boldsymbol{W}_0, \boldsymbol{W}_1, \ldots, \boldsymbol{W}_L).$$

Here the $\boldsymbol{W}$'s are vectors or matrices, "$\circ$" means that the scalar function is applied to each component of the vector. In practice, it has been found that training such networks is often quite hard when L is large due to the *exploding or vanishing gradient problem* [50]: The gradient with respect to the parameters either grows or diminishes fast as L, the number of layers or the depth, increases. This problem is very much alleviated if one switches to a residual form:

$$\begin{aligned} \boldsymbol{z}_0(\boldsymbol{x}) &= \boldsymbol{V}\boldsymbol{x}, \\ \boldsymbol{z}_{l+1}(\boldsymbol{x}) &= \boldsymbol{z}_l(\boldsymbol{x}) + \boldsymbol{U}_l \sigma \circ \big(\boldsymbol{W}_l \boldsymbol{z}_l(\boldsymbol{x})\big), \quad l = 0, 1, \ldots, L-1, \end{aligned} \tag{1.1}$$

and $f(\boldsymbol{x}, \theta) = \alpha \cdot \boldsymbol{z}_L(\boldsymbol{x})$ for some vector α. This is the *residual neural network* model, or the *ResNet* model [48]. The issue of exploding or vanishing gradients has been analyzed for DNNs in [47], but we still lack a rigorous mathematical analysis for ResNets.

In addition to supervised learning, there are two other major subjects in classical machine learning:

- unsupervised learning, which is mainly concerned with finding some aspects of an underlying probability distribution using a finite sample;
- reinforcement learning, which is about finding the optimal strategy for a *Markov decision process* [93].

Deep neural network-based ML is commonly referred to as *deep learning* [62, 85].

Deep learning is a very powerful tool. In the last ten years or so, deep learning has achieved tremendous success for a wide variety of problems. The most representative example is in computer vision, e.g., the classification of images. Typically, the images are labeled into several different categories according to the content of each image. Our task is to predict the correct category for images of the same kind. This is a supervised learning problem where the target function is the mapping from each image to its content, i.e., the category of that image.[1]

Another example is generating extremely real-looking pictures of fake human faces.[2] Using pictures of real human faces as samples, generative ML models can produce new samples which are pictures of fake human faces. This is an example of unsupervised learning. We can view pictures of human faces as being a random variable in the spaces of images. The probability distribution of that random variable is unknown to us. But we do know some samples of that probability distribution, namely the pictures of real human faces. From that sample, one can approximate the underlying probability distribution sufficiently accurately that one can produce new samples. These new samples are the pictures of fake human faces.

The best known example of reinforcement learning is AlphaGo [90]. Given the strategy of the opponent, the Go game can be formulated as a Markov decision process whose optimal strategy satisfies the underlying Bellman equation. What AlphaGo did was to solve that Bellman equation approximately for an increasingly better opponent.

Approximating functions, probability distributions, and solutions of difference or differential equations are among the most common tasks in computational mathematics. One is naturally led to ask: What is different in the tasks described above from those that are commonly done in mathematics? One most important difference is the dimensionality of the problems. Take the CIFAR-10 dataset as an example. We can view each image as being a point in a $d = 32 \times 32 \times 3 = 3072$-dimensional space, counting the number of pixels and the dimensionality of the color space. Classical algorithms in computational mathematics are not able to handle problems in such high dimension.

The curse of dimensionality. To see this more clearly, let us take a look at a typical result in classical approximation theory, the approximation by piecewise linear functions over a regular mesh. Let h be the typical size of the mesh. Then we have

$$\inf_{f \in \mathcal{H}_m} \left\| f^* - f \right\|_{L^2(X)} \leq C_d h^2 \left\| f^* \right\|_{H^2(X)} \sim C_d m^{-2/d} \left\| f^* \right\|_{H^2(X)},$$

where $\| f^* \|_{H^2(X)}$ is the Sobolev norm of f^*. If we want to reduce the error by a factor of 10, we need to reduce h by a factor of $\sqrt{10}$ and increase m by a factor of $10^{d/2}$. For $d = 3072$, this is truly a huge number.

1 See, for example, https://www.cs.toronto.edu/~kriz/cifar.html.

2 See, for example, https://machinelearningmastery.com/resources-for-getting-started-with-generative-adversarial-networks/.

The problem described here is referred to as the *curse of dimensionality* (CoD): As dimensionality grows, computational cost grows exponentially. This phenomenon is common to all classical algorithms, such as algorithms based on fixed meshes and wavelets.

CoD has been a major obstacle for many problems in science and engineering, including quantum and classical many-body problems, dynamic programming and control problems, and nonparametric statistics. Before deep learning, many approximate algorithms and models have been developed to bypass the CoD problems. The most well-known ones include the Hartree and Hartree–Fock approximation in quantum mechanics, the generalized linear models in statistics, and approximate dynamic programming models. Although these models are heavily used in practice, we lack systematic ways to improve their accuracy. It is fair to say that deep learning seems to be the first general methodology that is capable of handling a large class of such problems with satisfactory accuracy.

2. DEEP LEARNING-BASED ALGORITHMS FOR PROBLEMS IN SCIENTIFIC COMPUTING

Deep learning has been very successful for many high-dimensional problems in computer vision and natural language processing [62]. It is natural to ask whether it can be used to solve high-dimensional problems in other areas such as scientific computing and computational science. This has indeed been a very active research area since 2016. Below we briefly review some representative progresses in this direction.

2.1. Control problems

The first successful application of deep learning to problems in scientific computing was presented in [43] for stochastic control problems. Consider the stochastic dynamic model

$$\boldsymbol{z}_{l+1} = \boldsymbol{z}_l + \mathbf{g}_l(\boldsymbol{z}_l, \boldsymbol{u}_l) + \xi_l, \tag{2.1}$$

where $\boldsymbol{z}_l$, $\boldsymbol{u}_l$, ξ_l denotes the state of the system, the control, and the noise at step l, respectively. Our objective is

$$\min_{\{\boldsymbol{u}_l\}_{l=0}^{T-1}} \mathbb{E}_{\{\xi_l\}} \left\{ \sum_{l=0}^{T-1} c_l\big(\boldsymbol{z}_l, \boldsymbol{u}_l(\boldsymbol{z}_l)\big) + c_T(\boldsymbol{z}_T) \right\}. \tag{2.2}$$

We are interested in looking for the feedback control (or closed-loop control)

$$\boldsymbol{u}_l = \boldsymbol{u}_l(\boldsymbol{z}),$$

and we will approximate this function by some neural network model (the details of the network model is not important for this discussion)

$$\boldsymbol{u}_l(\boldsymbol{z}) \approx \tilde{\boldsymbol{u}}_l(\boldsymbol{z}|\theta_l), \quad l = 0, \ldots, T-1.$$

With this approximation, the optimization problem becomes

$$\min_{\{\theta_l\}_{l=0}^{T-1}} \mathbb{E}_{\{\xi_l\}} \left\{ \sum_{l=0}^{T-1} c_l\big(\boldsymbol{z}_l, \tilde{\boldsymbol{u}}_l(\boldsymbol{z}_l|\theta_l)\big) + c_T(\boldsymbol{z}_T) \right\}. \tag{2.3}$$

This was the first example on developing deep learning-based algorithms for problems in scientific computing. The motivation for using this as the first example was the close similarity between stochastic control problems and ResNet-based deep learning: the dynamic model (2.1) in the control problem plays the role of the ResNet, the objective function (2.2) plays the role of the empirical risk and the random noise in (2.1) plays the role of the training data. Using this analogy, Han and E developed an SGD and neural network-based algorithm for the stochastic control problem and demonstrated that it can readily handle very high dimensional problems [43]. The neural network model used was a composite network, with the control at each step represented by a subnetwork.

Subsequently, there have been many developments on deep learning-based algorithms for control problems. For a survey of the activities in this area, we refer to the UCLA IPAM workshop in the Spring of 2020. We mention in particular the extension to deterministic control problems in [77]. These developments have demonstrated adequately the potential of deep learning-based algorithms for solving real world control problems. Yet there are still serious work to be done to fully realize that potential in practice. There are two main obstacles. The first is that we often lack reliable dynamic models for the practical problems we are interested in. The second is the robustness of the deep learning-based algorithms in realistic settings.

2.2. High-dimensional partial differential equations

Motivated by the success for control problems, E, Han, and Jentzen developed deep learning-based algorithms for nonlinear parabolic partial differential equations (PDEs). The idea is to use backward stochastic differential equations (BSDEs) to reformulate the nonlinear PDE as a control-like problem, and then follow similar strategies for stochastic control problems [26, 44].

Consider the initial value problem

$$\frac{\partial v}{\partial t} = \frac{1}{2}\sigma\sigma^T : \nabla^2 v + \mu \cdot \nabla v + f(\sigma^{\mathrm{T}}\nabla v), \quad v(0, \boldsymbol{x}) = g(\boldsymbol{x}).$$

It is better to turn this into a terminal value problem by reversing the direction of time. Let $u(t, \cdot) = v(T - t, \cdot)$. Then the problem above becomes

$$\frac{\partial u}{\partial t} + \frac{1}{2}\sigma\sigma^T : \nabla^2 u + \mu \cdot \nabla u + f(\sigma^{\mathrm{T}}\nabla u) = 0, \quad u(T, \boldsymbol{x}) = g(\boldsymbol{x}).$$

One can reformulate this as a stochastic optimization problem using BSDEs [78]:

$$\inf_{Y_0, \{Z_t\}_{0\le t\le T}} \mathbb{E}\left|g(X_T) - Y_T\right|^2, \tag{2.4}$$

$$\text{such that} \quad X_t = X_0 + \int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dW_s, \tag{2.5}$$

$$Y_t = Y_0 - \int_0^t f(Z_s)\, ds + \int_0^t (Z_s)^{\mathrm{T}}\, dW_s. \tag{2.6}$$

It can be shown that both problems have unique solutions and these solutions are related to each other by [79]

$$Y_t = u(t, X_t) \quad \text{and} \quad Z_t = \sigma^T(t, X_t)\,\nabla u(t, X_t). \tag{2.7}$$

Problem (2.4) is very much like a stochastic control problem and one can then develop algorithms using ideas similar to those described above. The resulted algorithm, the *Deep BSDE method*, has turned out to be an elegant and powerful tool for solving (nonlinear) Black–Scholes equations in finance, Hamilton–Jacobi–Bellman equations, as well as BSDEs. See [27] for a review.

In the Deep BSDE method, much effort has gone into the reformulation of the PDE problem as a control-like problem, in order to explore the intrinsic structure of the underlying problem. In the opposite direction, [82,91] developed strategies that are "foolproof." The idea is to use least squares and formulate the PDE and boundary condition as an optimization problem, and then more or less blindly apply ML to that optimization problem [82,91]. This has become quite popular in applied mathematics since it offers applied mathematicians a way to gain experience in deep learning by playing with the problems they are familiar with.

2.3. Parametrizing solutions of differential equations

Another idea is to explore the representative power of deep neural network models and parametrize solutions of PDEs as a functional of the coefficients and boundary data. This was first demonstrated by Khoo, Lu, and Ying for the Schrödinger equation with random potential [59]. For a more systematic development along this direction, we refer to [66].

In contrast to most other applications in which the object of interest is a function (though maybe in high dimension), in this setting, the object of interest is an operator on an infinite-dimensional space. This raises new mathematical issues beyond those discussed below.

2.4. Molecular dynamics

In molecular dynamics, we model the dynamic trajectory of each atom in a material or a molecule by solving the Newton's equation

$$m_i \frac{d^2 \boldsymbol{x}_i}{dt^2} = -\nabla_{\boldsymbol{x}_i} V, \quad V = V(\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_i, \ldots, \boldsymbol{x}_N),$$

where $m_i, \boldsymbol{x}_i$ are the mass and position of the ith atom, respectively. The key question is how to model the potential energy function V that describes the interaction between the atoms. Traditionally, this has been modeled either empirically or by solving quantum mechanics-based models, such as density functional theory, on the fly computing the forces between the atoms [12,22]. Neither is satisfactory: the empirical approach is unreliable; the on-the-fly quantum mechanics-based approach is expensive and limited to systems with only hundreds or thousands of atoms.

With the advent of ML, we can contemplate a new paradigm in which quantum mechanics models are used to provide data, from which one can learn a highly accurate potential energy function, which can then be used to perform molecular dynamics. Such a paradigm was first proposed in [9]. One of the most successful examples of such a model is the Deep Potential models developed in [45,110] (see Figure 1). Using high performance computing resources, one can perform molecular dynamics calculation with *ab initio* accuracy for systems with hundreds of millions atoms [54].

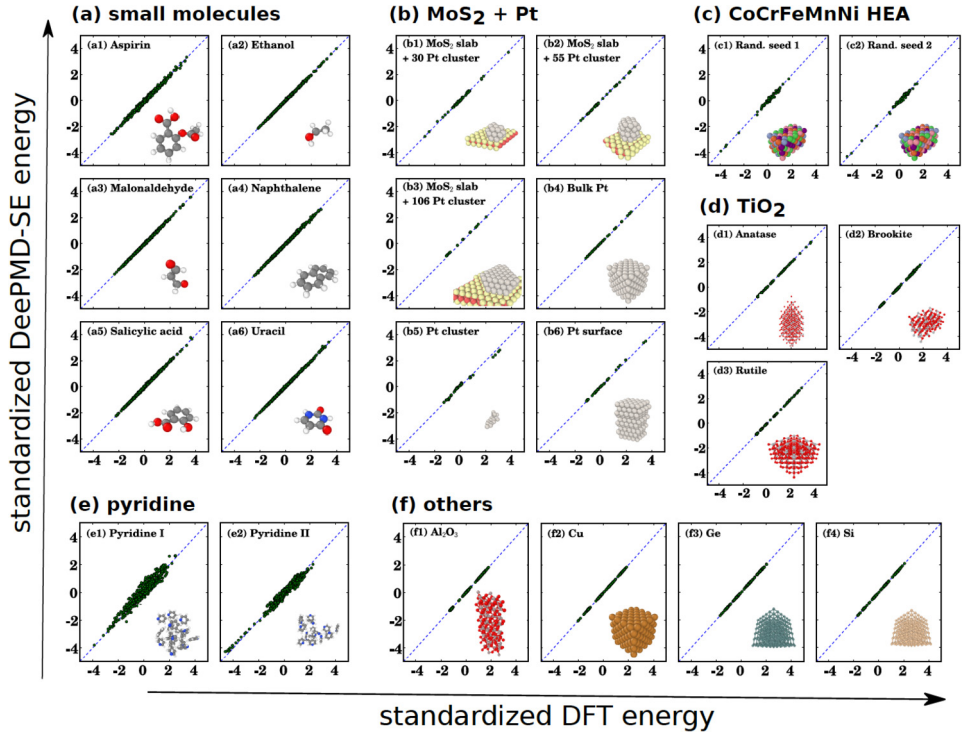

FIGURE 1

Comparison of the accuracy of the energies predicted by the Deep Potential model and density functional theory for different kinds of systems [110].

With the Deep Potential model, one can do many things that were either impossible or very difficult before. Examples include complex reaction processes in combustion [109], crystal nucleation of liquid silicon [10], liquid–liquid phase transition of water [38], one-dimensional cooperative diffusion in a three-dimensional crystal [98], structural order in quasicrystal growth [42], and the phase diagram of water [112].

2.5. Multiscale modeling

It has long been recognized that multiscale modeling can be a very effective tool in computational science and engineering (see Figure 2). However, its practical usage has been hampered by our inadequate ability to analyze the data obtained from the underlying microscopic model [22]. This is exactly where ML can help. Indeed ML-based *ab initio* molecular dynamics is an example of the application of ML to multiscale modeling. Besides molecular dynamics, ML-based multiscale models have been developed for density functional theory, coarse-grained molecular dynamics, moment closure models for the kinetic equations, hydrodynamic models for non-Newtonian fluids, etc. There is no doubt that this will continue to be a very fruitful line of research.

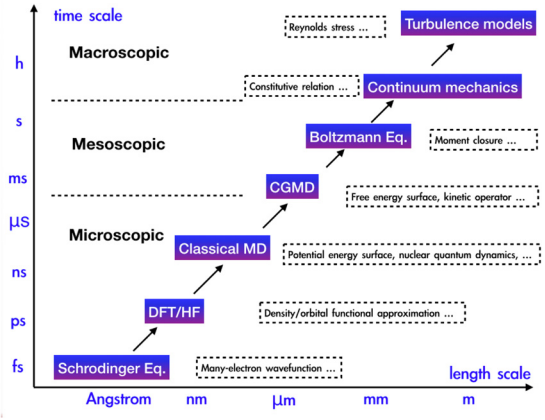

FIGURE 2
The hierarchy of multiscale physical models.

Most if not all existing applications of ML to multiscale modeling belong to the class of *sequential multiscale modeling* [22], i.e., ML algorithms are used at the pre-computing stage to obtain accurate coarse-grained models. One is naturally led to ask: How can we develop reliable and interpretable new physical models using ML? There are three most important issues involved here [28]:

- The first is how to collect the training data. The training dataset needs to be representative enough of all the practical situations that the model is intended for, yet at the same time, it needs to be as small as possible since each data point usually involves solving the microscale model. For this purpose, Zhang et al. developed the ELT (exploration–labeling–training) algorithm and it has been successfully applied to molecular dynamics and coarse-grained molecular dynamics [28, 111, 113].

- The second issue is the starting point of the new physical model. To be interpretable, it usually helps to formulate the new physical model as some kind of projection of the underlying microscale model. An example is the moment-closure model for the kinetic equation. The projection scheme, or the coarse-grained model, should not violate the physical conservation laws in the problem.
  To formulate this projection scheme, one needs to know the set of coarse-grained variables. In principle, ML can also be a very powerful tool for this purpose. In practice, this is still a relatively unexplored area.

- The projected model is usually not closed and involves terms that need to be modeled. These terms are analogous to the constitutive relations in classical models such as the Navier–Stokes equation. This situation is very similar to that of the heterogeneous multiscale method [22, 25]. The third issue is therefore to formulate

ML models for the unclosed terms in the coarse-grained model. To do so, one has to take into account physical constraints such as the symmetries in the system.

For a discussion of these issues, we refer to [28].

2.6. The many-electron Schrödinger equation

The many-electron Schrödinger equation in quantum mechanics is a notoriously hard problem not only due to its high dimensionality but also the fact that its solution must satisfy the Pauli exclusion principle, i.e., the wave-function must be antisymmetric. It is also arguably the most fundamental problem in computational science since it represents the true first principle. This latter feature is becoming increasingly more clear due to the advance of ML-based algorithms and the Schrödinger equation is the ultimate provider of the data that we use to train more coarse-grained models, particularly density functional theory models.

For the spin Schrödinger equation, Carleo and Troyer developed an algorithm using the restricted Boltzmann machine and the least squares formulation [13]. Deep learning-based algorithm for the many-electron Schrödinger equation was first developed in [46]. More sophisticated ansatz for the antisymmetric part of the wave-function was developed in [49,80]. A spectral projection algorithm was proposed in [102] to fully take advantage of the linear character of the Schrödinger equation. It is fair to say that at this stage, deep learning-based algorithm still remains an experimental effort and has not outperformed traditional quantum chemistry methods.

2.7. Purely data-driven methods

The most remarkable example of the purely data-driven method is AlphaFold2 [58]. By using only the structures in the protein data bank and protein sequence data, AlphaFold2 is able to predict the native structure of proteins to experimental accuracy. This was quite unexpected and has changed the way things are done in structural biology.

As a structural optimization problem, protein folding can be considered as a (classical) many-body problem. This is NOT how AlphaFold2 solved the problem. AlphaFold2 did not try to find the native structure by exploring the high-dimensional configuration space of the protein of some energy function. Instead, it took an interpolation viewpoint: Given the structures we know in the protein data bank, try to find the unknown structures by exploring the similarity between the given protein sequence and the sequences in the protein data bank. For this purpose, one needs to explore the structure of the sequence space. This is done by pushing multiple sequence alignment to a limit [58].

3. MATHEMATICAL THEORY OF NEURAL NETWORK-BASED MACHINE LEARNING MODELS

At this point, the objective of a mathematical theory for deep learning is not to explain in detail everything we see in practice, but rather to formulate general principles that can help organize our thoughts and guide future work.

The two most important puzzles in deep learning are:

- Why do deep learning models work so well on such seemingly very complicated tasks?
- Why does the performance of deep learning models depend so sensitively on the choice of the hyper-parameters, such as the network size, architecture, and the learning rate in the optimization algorithm?

A more advanced question is whether we can come up with new formulations of ML models that are both accurate and robust.

There are many different ways of looking at these issues, ranging from classical learning theory [94], statistical physics perspective [108], to information theory perspective [1]. We will take the viewpoint of classical numerical analysis (approximation theory, convergence of training algorithms, convergence rates, etc.) but put emphasis on the feature of high dimensionality. For a review along this line of thoughts, we refer to [34].

Before proceeding further, let us note that we will use the terminology "norm" in a loose way, in the sense that the triangle inequality is not necessarily satisfied.

3.1. An overview of approximation theory

Approximation theory is concerned with the question whether a given hypothesis space can efficiently approximate the target functions we are interested in. In this direction, there are three kinds of results.

The first is the so-called *Universal Approximation Theorem (UAT)*, which, roughly speaking, asserts that under mild conditions, one can use neural network functions to approximate arbitrary continuous functions uniformly on compact domains [18]. Such results are of course important, without them the whole foundation of neural network models would be in doubt, but they do not explain why neural network models are so much better than classical polynomial approximations. After all, as we know from the Weierstrass theorem, UAT also holds for polynomial approximations, which we know is a bad idea in high dimension. To see the difference between the two kinds of approximations, we must study the rate of convergence.

The second kind of results are convergence rates of neural network approximations for functions with certain regularity conditions. A typical result states that if a function has derivatives of order up to k, then it can be approximated by neural networks with an error of $O(m^{-k/d})$ where m is the total number of parameters. The first systematic result of this type can be found in [107]. The most recent and sharpest results can be found in [70]. These results do suffer from CoD. But they are useful for analyzing neural network-based algorithms for low dimensional problems.

The third kind of results are convergence rates for neural network approximations that do not suffer from CoD. This line of research began with the pioneering work of Barron [7, 8, 11, 57]. We will focus on this type of results.

3.2. General remarks about high-dimensional problems

Before continuing, let us recap the important parameters that we have: m is the dimensionality of the hypothesis space; n is the size of the training sample; d is the dimensionality of the input variable to the ML model. We are interested in the case when $d \gg 1$.

The one high-dimensional problem that has been very well studied is high dimensional numerical integration. We are interested in approximating the following integral:

$$I(g) = \int_X g(\boldsymbol{x}) d\boldsymbol{x}$$

by a sum $I_m(g) = \frac{1}{m}\sum_j g(\boldsymbol{x}_j)$. If we use grid-based quadrature rules such as the Trapezoidal Rule, then the error behaves like

$$I(g) - I_m(g) \sim \frac{C(g)}{m^{\alpha/d}}$$

for some fixed constant α, indicating CoD. If instead we use Monte Carlo integration, say by taking $\{\boldsymbol{x}_j, j \in [m]\}$ to be independent, uniformly distributed in X, then we have

$$\mathbb{E}\big(I(g) - I_m(g)\big)^2 = \frac{\mathrm{var}(g)}{m}, \quad \mathrm{var}(g) = \int_X g^2(\boldsymbol{x}) d\boldsymbol{x} - \left(\int_X g(\boldsymbol{x}) d\boldsymbol{x}\right)^2.$$

The $O(1/\sqrt{m})$ rate is (almost) the best we can hope for, and is independent of d: Improvements on the convergence rate, say using quasi-Monte Carlo or other lattices, diminish quickly as d becomes large [20].

The variance $\mathrm{var}(g)$ can be very large in high dimension. For this reason many variance-reduction algorithms have been developed. These ideas allow physicists to study statistical physical models in very high dimension.

Function approximation is a harder problem than numerical integration. In light of the discussion above, the best we can hope for function approximation in high dimension are results of the following type:

$$\inf_{f \in \mathcal{H}_m} \mathcal{R}(f) = \inf_{f \in \mathcal{H}_m} \left\| f - f^* \right\|^2_{L^2(d\mu)} \lesssim \frac{\Gamma(f^*)}{m}.$$

The questions that we need we address are: Can this be true? Given a neural network model, say two-layer neural networks or ResNets, for what class of functions is this true? If true, what should the quantity $\Gamma(f^*)$ be?

3.3. Approximation theory for the random feature model

To explain the general philosophy, we will use the random feature model [81] as an illustration. Let $\phi(\cdot; \boldsymbol{w})$ denote some feature function parametrized by $\boldsymbol{w}$, e.g., $\phi(\boldsymbol{x}, \boldsymbol{w}) = \sigma(\boldsymbol{w}^T \boldsymbol{x})$. A random feature model is defined by

$$f_m(\boldsymbol{x}; \boldsymbol{a}) = \frac{1}{m}\sum_{j=1}^m a_j \phi\big(\boldsymbol{x}; \boldsymbol{w}_j^0\big), \tag{3.1}$$

where $\{\boldsymbol{w}_j^0\}_{j=1}^m$ are i.i.d. samples drawn from a prefixed distribution π_0. Once drawn, $\{\boldsymbol{w}_j^0\}_{j=1}^m$ are fixed; $\boldsymbol{a} = (a_1, \ldots, a_m)^T \in \mathbb{R}^m$ are the trainable parameters. For simplicity, we assume $\Omega := \mathrm{supp}(\pi_0)$ is compact. Denote $W^0 = (\boldsymbol{w}_1^0, \ldots, \boldsymbol{w}_m^0)^T \in \mathbb{R}^{m \times d}$. Note that random feature models are linear models.

If the inner parameters $\{\boldsymbol{w}_j^0\}$ are also allowed to change, then this becomes a (generalized) two-layer neural network model. For this reason, the random feature model can be considered as a simplified two-layer neural network model in which the inner parameters are frozen at some random initial value. This connection has proven to be important for understanding the two-layer neural network model.

We are interested in identifying the function class and the functional $\Gamma(\cdot)$ for this model. To this end, consider the reproducing kernel Hilbert space (RKHS) [2] induced by the kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \mathbb{E}_{\boldsymbol{w}\sim\pi_0}[\phi(\boldsymbol{x};\boldsymbol{w})\phi(\boldsymbol{x}';\boldsymbol{w})]$. Denote by $\mathcal{H}_k$ this RKHS. Then for any $f \in \mathcal{H}_k$, there exists $a(\cdot) \in L^2(\pi_0)$ such that

$$f(\boldsymbol{x}) = \int a(\boldsymbol{w})\phi(\boldsymbol{x};\boldsymbol{w})d\pi_0(\boldsymbol{w}) \tag{3.2}$$

and

$$\|f\|_{\mathcal{H}_k}^2 = \int a^2(\boldsymbol{w})d\pi_0(\boldsymbol{w}). \tag{3.3}$$

Theorem 1 (Direct Approximation Theorem). *For any $f^* \in \mathcal{H}_k$, let*

$$f^*(\boldsymbol{x}) = \int a^*(\boldsymbol{w})\phi(\boldsymbol{x};\boldsymbol{w})d\pi_0(\boldsymbol{w}). \tag{3.4}$$

Then we have

$$\mathbb{E}_{\boldsymbol{W}^0}\left\|f_m\left(\cdot;a^*(\boldsymbol{W}^0)\right) - f^*\right\|_{L^2}^2 \le \frac{\|f^*\|_{\mathcal{H}_k}^2}{m},$$

where $a^*(\boldsymbol{W}^0) = (a^*(\boldsymbol{w}_1^0), \ldots, a^*(\boldsymbol{w}_m^0))^T$.

Theorem 2 (Inverse Approximation Theorem). *Let $(\boldsymbol{w}_j^0)_{j=0}^\infty$ be a realization of the sequence of i.i.d. random samples drawn from π_0. Let f^* be a continuous function on $X = [0,1]^d$. Assume that there exists a constant C and a sequence $(a_j)_{j=0}^\infty$ satisfying $\sup_j |a_j| \le C$, such that*

$$\lim_{m\to\infty} \frac{1}{m}\sum_{j=1}^m a_j\phi\left(\boldsymbol{x};\boldsymbol{w}_j^0\right) = f^*(\boldsymbol{x}), \tag{3.5}$$

for all $\boldsymbol{x} \in X$. Then with probability 1, there exists a function $a^*(\cdot) : \Omega \mapsto \mathbb{R}$ such that

$$f^*(\boldsymbol{x}) = \int_\Omega a^*(\boldsymbol{w})\phi(\boldsymbol{x};\boldsymbol{w})d\pi_0(\boldsymbol{w}).$$

Moreover, $\|a^*\|_\infty \le C$.

This pair of direct and inverse theorems are not exactly converses of each other since different norms (L^2 and L^∞) are used. But they do tell us that the associated RKHS is the appropriate function space to study in connection with the random feature model. These results are not new, but it seems difficult to identify the origin of these results.

3.4. Approximation theory for two-layer neural networks

We will restrict our attention to the case when ReLU is used as the activation function. The hypothesis space for two-layer neural networks is defined by

$$\mathcal{H}_m = \left\{ f_m(\boldsymbol{x}) = \frac{1}{m} \sum_j a_j \sigma\left(\boldsymbol{w}_j^T \boldsymbol{x}\right) \right\}$$

A good candidate for the associated function space for this model is the *Barron space* [30,33] (see also [6,7,35,60]). To define the Barron space, consider functions $f : X = [0,1]^d \mapsto \mathbb{R}$ of the following form:

$$f(\boldsymbol{x}) = \int_\Omega a\sigma(\boldsymbol{w}^T \boldsymbol{x})\rho(da, d\boldsymbol{w}) = \mathbb{E}_\rho\big[a\sigma(\boldsymbol{w}^T \boldsymbol{x})\big], \quad \boldsymbol{x} \in X,$$

where $\Omega = \mathbb{R}^1 \times \mathbb{R}^{d+1}$ and ρ is a probability distribution on Ω. The "Barron norm" is defined by

$$\|f\|_{\mathcal{B}_p} = \inf_{\rho \in P_f} \left(\mathbb{E}_\rho\big[a^p \|\boldsymbol{w}\|_1^p\big]\right)^{1/p},$$

where $P_f := \{\rho : f(\boldsymbol{x}) = \mathbb{E}_\rho[a\sigma(\boldsymbol{w}^T \boldsymbol{x})]\}$. Let $\mathcal{B}_p = \{f \in C^0 : \|f\|_{\mathcal{B}_p} < \infty\}$. Functions in $\mathcal{B}_p$ are called Barron functions. As was shown in [33], we actually have $\|\cdot\|_{\mathcal{B}_p} = \|\cdot\|_{\mathcal{B}_q}$ for any $1 \le p \le q \le \infty$. Hence, we will use $\|\cdot\|_{\mathcal{B}}$ and $\mathcal{B}$ denote the Barron norm and Barron space.

One immediate question is: What kinds of function are Barron functions? In this direction, a general result is given by:

Theorem 3 ([60]). *Let $\gamma_2(f) := \int_{\mathbb{R}^d} \|\omega\|_1^2 |\tilde{f}(\omega)| d\omega < \infty$, where $\tilde{f}$ is the Fourier transform of f, then f can be represented as*

$$f(\boldsymbol{x}) = \int_\Omega a\sigma(\boldsymbol{w}^T \boldsymbol{x})\rho(da, d\boldsymbol{w}).$$

Moreover, $\|f\|_{\mathcal{B}} \le 2\gamma_2(f) + 2\|\nabla f(0)\|_1 + 2|f(0)|$.

Remark 4. One should note the difference between the Barron norm and the quantities like γ_2 which were originally introduced by Barron [7]. The Barron norm is defined using a probabilistic setting. The quantity $\gamma_2(f)$ and the like are defined using the Fourier transform which is related to the regularity property of f. We believe that the probabilistic setup is the right direction to go and this is partly confirmed by subsequent results on continuous ResNets (see below) and multilayer neural networks. To avoid further confusion, we propose to call quantities γ_2 and the like *Barron's spectral norm*. For further results in this direction, as well as some interesting analysis on the relationship between these spaces, we refer to [88,89].

An interesting structural theorem about Barron functions is proved in [35].

Theorem 5. *Let f be in Barron space. Then $f = \sum_{i=1}^\infty f_i$ where $f_i \in C^1(X \setminus V_i)$ where V_i is a k-dimensional affine subspace of X for some $0 \le k \le d-1$.*

As an immediate corollary, we see that the distance to the unit sphere is not a Barron function.

The claim that Barron space is the natural space associated with two-layer networks is justified by the following series of results [30].

Theorem 6 (Direct Approximation Theorem). *For any $f \in \mathcal{B}$ and $m \in \mathbb{N}^+$, there exists a two-layer neural network f_m with m neurons $\{(a_i, \boldsymbol{w}_i)\}$ such that*

$$\|f - f_m\|_{L^2(P)} \lesssim \frac{\|f\|_{\mathcal{B}}}{\sqrt{m}}.$$

Theorem 7 (Inverse Approximation Theorem). *Let*

$$\mathcal{N}_C \overset{\text{def}}{=} \left\{ \frac{1}{m} \sum_{k=1}^{m} a_k \sigma\left(\boldsymbol{w}_k^T \boldsymbol{x}\right) : \frac{1}{m} \sum_{k=1}^{m} |a_k| \|\boldsymbol{w}_k\|_1 \le C,\ m \in \mathbb{N}^+ \right\}.$$

Let f^* be a continuous function. Assume there exist a constant C and a sequence of functions $\{f_m\} \subset \mathcal{N}_C$ such that

$$f_m(\boldsymbol{x}) \to f^*(\boldsymbol{x})$$

for all $\boldsymbol{x} \in X$, then there exists a probability distribution ρ^* on Ω such that

$$f^*(\boldsymbol{x}) = \int a\sigma(\boldsymbol{w}^T \boldsymbol{x}) \rho^*(da, d\boldsymbol{w}),$$

for all $\boldsymbol{x} \in X$. Moreover, $\|f^*\|_{\mathcal{B}} \le C$.

3.5. Approximation theory for residual neural networks

Consider a residual network model

$$\begin{aligned} \boldsymbol{z}_0(\boldsymbol{x}) &= \boldsymbol{V}\boldsymbol{x}, \\ \boldsymbol{z}_{l+1}(\boldsymbol{x}) &= z_l(\boldsymbol{x}) + \frac{1}{L} \boldsymbol{U}_l \sigma \circ \left(\boldsymbol{W}_l z_l(\boldsymbol{x})\right), \quad l = 0, 1, \ldots, L-1, \\ f(\boldsymbol{x}, \theta) &= \alpha \cdot \boldsymbol{z}_L(\boldsymbol{x}), \end{aligned}$$

where $\boldsymbol{x} \in \mathbb{R}^d$ is the input, $\boldsymbol{V} \in \mathbb{R}^{D \times d}$, $D \ge d$, $\boldsymbol{W}_l \in \mathbb{R}^{m \times D}$, $\boldsymbol{U}_l \in \mathbb{R}^{D \times m}$, $\alpha \in \mathbb{R}^D$ and we use $\Theta := \{\boldsymbol{V}, \boldsymbol{U}_1, \ldots, \boldsymbol{U}_L, \boldsymbol{W}_l, \ldots, \boldsymbol{W}_L, \alpha\}$ to denote all the parameters to be learned from data. Without loss of generality, we will fix $\boldsymbol{V}$ to be

$$\boldsymbol{V} = \begin{bmatrix} \boldsymbol{I}_{d \times d} \\ \boldsymbol{0}_{(D-d) \times d} \end{bmatrix}. \tag{3.6}$$

To look for the appropriate associated function space, let us consider the following flow-based representation of functions (see next section):

$$\begin{aligned} \boldsymbol{z}(\boldsymbol{x}, 0) &= \boldsymbol{V}\boldsymbol{x}, \\ \dot{\boldsymbol{z}}(\boldsymbol{x}, t) &= \mathbb{E}_{(\boldsymbol{U}, \boldsymbol{W}) \sim \rho_t} \boldsymbol{U} \sigma\left(\boldsymbol{W}\boldsymbol{z}(\boldsymbol{x}, t)\right), \\ f_{\alpha, \{\rho_t\}}(\boldsymbol{x}) &= \alpha^T \boldsymbol{z}(\boldsymbol{x}, 1). \end{aligned}$$

For $p \ge 1$, consider the following linear ODEs associated with the representation above:

$$\begin{aligned} N_p(0) &= \mathbf{e}, \\ \dot{N}_p(t) &= \left(\mathbb{E}_{\rho_t}\left(|\boldsymbol{U}||\boldsymbol{W}|\right)^p\right)^{1/p} N_p(t), \end{aligned}$$

where $\boldsymbol{e}$ is a column vector with every component equal to 1, $|\mathbf{A}|$ and $\mathbf{A}^q$ are elementwise operations for the matrix $\mathbf{A}$ and a positive number q. The following function spaces and "norms" were introduced in [33].

Definition 8. Let f be a function that satisfies $f = f_{\alpha,\{\rho_t\}}$ for a pair of $(\alpha, \{\rho_t\})$. Define

$$\|f\|_{\mathcal{D}_p(\alpha,\{\rho_t\})} = |\alpha|^T N_p(1) \tag{3.7}$$

to be the $\mathcal{D}_p$ norm of f with respect to the pair $(\alpha, \{\rho_t\})$, where $|\alpha|$ is a vector obtained from α by taking elementwise absolute values. We define

$$\|f\|_{\mathcal{D}_p} = \inf_{f=f_{\alpha,\{\rho_t\}}} |\alpha|^T N_p(1) \tag{3.8}$$

to be the $\mathcal{D}_p$ norm of f, and let $\mathcal{D}_p = \{f : \|f\|_{\mathcal{D}_p} < \infty\}$.

Definition 9. Let f be a function that satisfies $f = f_{\alpha,\{\rho_t\}}$ for a pair of $(\alpha, \{\rho_t\})$. Define

$$\|f\|_{\tilde{\mathcal{D}}_p(\alpha,\{\rho_t\})} = |\alpha|^T N_p(1) + \left\|N_p(1)\right\|_1 - D \tag{3.9}$$

to be the $\tilde{\mathcal{D}}_p$ norm of f with respect to the pair $(\alpha, \{\rho_t\})$. We define

$$\|f\|_{\tilde{\mathcal{D}}_p} = \inf_{f=f_{\alpha,\{\rho_t\}}} |\alpha|^T N_p(1) + \left\|N_p(1)\right\|_1 - D \tag{3.10}$$

to be the $\tilde{\mathcal{D}}_p$ norm of f. The space $\tilde{\mathcal{D}}_p$ is defined as the set of functions that admit the representation $f_{\alpha,\{\rho_t\}}$ with finite $\tilde{\mathcal{D}}_p$ norm.

These two kinds of "norms" appear to be similar but different. These function spaces were introduced in [33] and are named *flow-induced function spaces*.

For the approximation theorems, we will make use of the following "Lipschitz continuity" condition for $\{\rho_t\}$.

Definition 10. Given a family of probability distributions $\{\rho_t,\ t \in [0,1]\}$, the "Lipschitz coefficient" of $\{\rho_t\}$, denoted by $\mathrm{Lip}_{\{\rho_t\}}$, is defined as the infimum of all the numbers that satisfy

$$\left|\mathbb{E}_{\rho_t} \boldsymbol{U}\sigma(\boldsymbol{W}\boldsymbol{z}) - \mathbb{E}_{\rho_s} \boldsymbol{U}\sigma(\boldsymbol{W}\boldsymbol{z})\right| \le \mathrm{Lip}_{\{\rho_t\}}|t-s||\boldsymbol{z}| \tag{3.11}$$

and

$$\left|\left\|\mathbb{E}_{\rho_t}|\boldsymbol{U}||\boldsymbol{W}|\right\|_{1,1} - \left\|\mathbb{E}_{\rho_s}|\boldsymbol{U}||\boldsymbol{W}|\right\|_{1,1}\right| \le \mathrm{Lip}_{\{\rho_t\}}|t-s|, \tag{3.12}$$

for any $t, s \in [0,1]$, where $\|\cdot\|_{1,1}$ is the sum of the absolute values of all the entries in a matrix. The "Lipschitz norm" of $\{\rho_t\}$ is defined as

$$\left\|\{\rho_t\}\right\|_{\mathrm{Lip}} = \left\|\mathbb{E}_{\rho_0}|\boldsymbol{U}||\boldsymbol{W}|\right\|_{1,1} + \mathrm{Lip}_{\{\rho_t\}}. \tag{3.13}$$

Finally, we define a discrete "path norm" for residual networks.

Definition 11. For a residual network defined by (3.6) with parameters $\Theta = \{\alpha, \boldsymbol{U}_l, \boldsymbol{W}_l, l = 0, 1, \ldots, L-1\}$, we define the l_1 path norm of Θ to be

$$\|\Theta\|_{\mathcal{P}} = |\alpha|^T \prod_{l=1}^{L}\left(I + \frac{1}{L}|\boldsymbol{U}_l||\boldsymbol{W}_l|\right)\boldsymbol{e}. \tag{3.14}$$

With the definitions above, we are ready to state the direct and inverse approximation theorems in the flow-induced function spaces [33].

Theorem 12 (Direct Approximation Theorem). *Let $f \in \tilde{\mathcal{D}}_2$, $\delta \in (0,1)$. Assume there exists a constant l_0 such that, for any $\varepsilon > 0$, there exists $(\alpha, \{\rho_t\})$ that satisfies $f = f_{\alpha,\{\rho_t\}}$ and $\|f\|_{\tilde{\mathcal{D}}_1(\alpha,\{\rho_t\})} < \|f\|_{\tilde{\mathcal{D}}_1} + \varepsilon$, $\|\{\rho_t\}\|_{\mathrm{Lip}} \le l_0$. Then there exists an L_0, depending polynomially on D, m, l_0, and $\|f\|_{\tilde{\mathcal{D}}_1}$, such that for any $L \ge L_0$, there exists an L-layer residual network $f_L(\cdot;\Theta)$ that satisfies*

$$\left\| f - f_L(\cdot;\Theta) \right\|^2 \le \frac{3\|f\|^2_{\tilde{\mathcal{D}}_1}}{L^{1-\delta}} \tag{3.15}$$

and

$$\|\Theta\|_{\mathcal{P}} \le 9\|f\|_{\tilde{\mathcal{D}}_1}. \tag{3.16}$$

Theorem 13 (Inverse Approximation Theorem). *Let f^* be a function defined on $X = [0,1]^d$. Assume that there exists a sequence of residual networks $\{f_L(\cdot;\Theta_L)\}_{L=1}^{\infty}$ such that $\|f^*(\boldsymbol{x}) - f_L(\boldsymbol{x};\Theta)\| \to 0$ as $L \to \infty$ for all $\boldsymbol{x} \in X$. Assume further that the parameters in $\{f_L(\cdot;\Theta)\}_{L=1}^{\infty}$ are (entrywise) bounded by c_0. Then, we have $f^* \in \mathcal{D}_\infty$ and $\|f^*\|_{\mathcal{D}_\infty} \le \frac{2e^{m(c_0^2+1)}D^2c_0}{m}$. Moreover, if there exists a constant c_1 such that $\|f_L\|_{\mathcal{D}_1} \le c_1$ holds for any $L > 0$, then we have $\|f^*\|_{\mathcal{D}_1} \le c_1$.*

A natural question is how big the flow-induced norms are compared with the Barron norm. In this direction, we have [33]

Theorem 14. *For any function $f \in \mathcal{B}$, and $D \ge d+2$, $m \ge 1$, we have $f \in \tilde{\mathcal{D}}_1$ and*

$$\|f\|_{\tilde{\mathcal{D}}_1} \le 2\|f\|_{\mathcal{B}}. \tag{3.17}$$

In this sense, going from two-layer neural networks to ResNets is like variance reduction in Monte Carlo methods.

3.6. The generalization gap

The second main issue in theoretical machine learning is the difference between training and test accuracy, in other words, the difference between the empirical and population risk. Estimating the difference between these two quantities is complicated by the fact that the parameters obtained from the training process are highly correlated with the data. There are many ways to bypass this difficulty. The simplest idea is to use the trivial bound

$$\left|\mathcal{R}(\hat{f}) - \mathcal{R}_n(\hat{f})\right| \le \sup_{f\in\mathcal{H}_m} \left|\mathcal{R}(f) - \mathcal{R}_n(f)\right| = \sup_{f\in\mathcal{H}_m} \left|I(g) - I_n(g)\right|, \tag{3.18}$$

where $\hat{f} \in \mathcal{H}_m$ is the function in the hypothesis space obtained from training, $g = (f - f^*)^2$. One of the most effective ways of estimating the right-hand side is to use the notion of Rademacher complexity.

Definition 15. Let $\mathcal{H}$ be a set of functions, and $S = (\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n)$ be a set of data points. The Rademacher complexity of $\mathcal{H}$ with respect to S is defined as

$$\mathrm{Rad}_S(\mathcal{H}) = \frac{1}{n}\mathbb{E}_\xi\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^n \xi_i h(\boldsymbol{x}_i)\right], \tag{3.19}$$

where $\{\xi_i\}_{i=1}^n$ are i.i.d. random variables taking values ± 1 with equal probability.

Rademacher complexity is useful since it bounds the quantity of interest, $\sup_{h\in\mathcal{H}} |I(h) - I_n(h)|$, from above and below.

Theorem 16 ([86, THEOREM 26.5]). *For any $\delta \in (0,1)$, with probability at least $1-\delta$ over the random samples $S = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$, we have*

$$\sup_{h\in\mathcal{H}}\left|\mathbb{E}_{\boldsymbol{x}}\big[h(\boldsymbol{x})\big] - \frac{1}{n}\sum_{i=1}^n h(\boldsymbol{x}_i)\right| \le 2\,\mathrm{Rad}_S(\mathcal{H}) + \sup_{h\in\mathcal{H}}\|h\|_\infty\sqrt{\frac{\log(2/\delta)}{2n}},$$

$$\sup_{h\in\mathcal{H}}\left|\mathbb{E}_{\boldsymbol{x}}\big[h(\boldsymbol{x})\big] - \frac{1}{n}\sum_{i=1}^n h(\boldsymbol{x}_i)\right| \ge \frac{1}{2}\,\mathrm{Rad}_S(\mathcal{H}) - \sup_{h\in\mathcal{H}}\|h\|_\infty\sqrt{\frac{\log(2/\delta)}{2n}}.$$

Roughly speaking, Rademacher complexity quantifies the degree to which functions in the hypothesis space can approximate random noise on a given dataset. The larger the hypothesis space, the larger the Rademacher complexity.

As example, if $\mathcal{H}$ is the unit ball in the space of continuous functions, then we obviously have $\mathrm{Rad}_S(\mathcal{H}) = O(1)$. If if $\mathcal{H}$ is the unit ball in the space of Lipschitz continuous functions, then it can be shown that [96]

$$\mathrm{Rad}_S(\mathcal{H}) = O(n^{-1/d}).$$

This signals another potential source of CoD, namely that the training sample size needed grows exponentially as d grows.

Fortunately, for the function spaces we identified earlier, their Rademacher complexity has roughly the optimal scaling. For Barron space, we have

Theorem 17 ([6]). *Let $\mathcal{F}_Q = \{f \in \mathcal{B}, \|f\|_{\mathcal{B}} \le Q\}$ and let $S = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$. Then we have*

$$\mathrm{Rad}_S(\mathcal{F}_Q) \le 2Q\sqrt{\frac{2\ln(2d)}{n}}.$$

The $n^{-1/2}$ scaling at the right-hand side is consistent with the Monte Carlo scaling that one would expect at a first sight.

The Rademacher complexity estimate is only established for a family of modified flow-induced function norms $\|\cdot\|_{\hat{\mathcal{D}}_p}$ (note the factor 2 in the definition below). It is not clear at this stage whether this is only a technical issue.

Let

$$\|f\|_{\hat{\mathcal{D}}_p} = \inf_{f=f_{\alpha,\{\rho_t\}}} |\alpha|^T \hat{N}_p(1) + \left\|\hat{N}_p(1)\right\|_1 - D + \left\|\{\rho_t\}\right\|_{\mathrm{Lip}}, \tag{3.20}$$

where $\hat{N}_p(t)$ is given by

$$\hat{N}_p(0) = 2\boldsymbol{e},$$
$$\dot{\hat{N}}_p(t) = 2\big(\mathbb{E}_{\rho_t}\big(|\boldsymbol{U}||\boldsymbol{W}|\big)^p\big)^{1/p}\hat{N}_p(t).$$

Denote by $\hat{\mathcal{D}}_p$ the space of functions with finite $\hat{\mathcal{D}}_p$ norm. Then, we have

Theorem 18 ([33]). *Let* $\hat{\mathcal{D}}_p^Q = \{f \in \hat{\mathcal{D}}_p : \|f\|_{\hat{\mathcal{D}}_p} \le Q\}$ *and let* $S = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_n)$. *Then we have*

$$\mathrm{Rad}_S\big(\hat{\mathcal{D}}_2^Q\big) \le 18Q\sqrt{\frac{2\log(2d)}{n}}. \tag{3.21}$$

3.7. A priori estimates of the population risk for regularized models

Our objective is to show that one can find accurate approximations of the target function using a finite training sample. Ideally, we would like to have the following kind of results:

$$\mathcal{R}(\hat{f}) \lesssim \frac{\Gamma_1(f^*)}{m} + \frac{\Gamma_2(f^*)}{\sqrt{n}}. \tag{3.22}$$

For appropriately regularized models, results of this kind have been established for the random feature model, the two-layer neural network model, and ResNets.

For the random feature model, consider the regularized model

$$\mathcal{L}_{n,\lambda}(\boldsymbol{a}) = \mathcal{R}_n(\boldsymbol{a}) + \frac{\lambda}{\sqrt{n}}\frac{\|\boldsymbol{a}\|}{\sqrt{m}},$$

and define the regularized estimator

$$\hat{\boldsymbol{a}}_{n,\lambda} = \operatorname{argmin} \mathcal{L}_{n,\lambda}(\boldsymbol{a}).$$

Theorem 19. *Fix any* $\lambda > 0$. *For any* $\delta \in (0,1)$, *with probability* $1-\delta$, *we have*

$$\begin{aligned}\mathcal{R}(\hat{\boldsymbol{a}}_{n,\lambda}) \le{}& \frac{1}{m}\left(\log(n/\delta)\|f^*\|_{\mathcal{H}_k}^2 + \frac{\log^2(n/\delta)}{m}\|a^*\|_\infty^2\right)\\ &+ \frac{1}{\sqrt{n}}\left(\|f^*\|_{\mathcal{H}_k} + \left(\frac{\log(1/\delta)}{m}\right)^{1/4}\|a^*\|_\infty + \sqrt{\log(2/\delta)}\right).\end{aligned} \tag{3.23}$$

Such results should be standard. But a complete proof seems to be only found in [34].

In the same way, for the two-layer neural network model, one can consider the regularized model

$$\mathcal{L}_n(\theta) = \mathcal{R}_n(\theta) + \lambda\sqrt{\frac{\log(2d)}{n}}\|\theta\|_{\mathcal{P}}, \quad \hat{\theta}_{n,\lambda} = \operatorname{argmin} \mathcal{L}_n(\theta),$$

where the path norm is defined by

$$\|\theta\|_{\mathcal{P}} = \frac{1}{m}\sum_{j=1}^m |a_j|\|\boldsymbol{w}_j\|_1,$$

and let $\hat{\theta}_n = \operatorname{argmin} \mathcal{L}_n(\theta)$.

Theorem 20 ([30]). *Assume $f^* : X \mapsto [0,1] \in \mathcal{B}$. There exists an absolute constant C_0 such that if $\lambda \geq C_0$ then for any $\delta > 0$, with probability at least $1-\delta$ over the choice of the training set, we have*

$$\mathcal{R}(\hat{\theta}_n) \lesssim \frac{\|f^*\|_{\mathcal{B}}^2}{m} + \lambda \|f^*\|_{\mathcal{B}} \sqrt{\frac{\log(2d)}{n}} + \sqrt{\frac{\log(n/\delta)}{n}}.$$

For ResNets, instead of the path norm (3.14), we have to consider a weighted path norm

$$\|\Theta\|_{WP} = |\alpha|^T \prod_{l=1}^{L} \left(I + \frac{2}{L} |U_l||W_l| \right) \boldsymbol{e}, \tag{3.24}$$

which assigns larger weights to paths that pass through more nonlinearities. Consider the regularized model

$$\min_{\Theta} \mathcal{J}(\Theta) = \hat{\mathcal{R}}(\Theta) + 3\lambda \|\Theta\|_{WP} \sqrt{\frac{2\log(2d)}{n}}. \tag{3.25}$$

Theorem 21 ([29]). *Let $f^* : X \to [0,1]$. Assume that $\widehat{\Theta}$ is an optimal solution of the regularized model (3.25). Let $\lambda \geq 4 + 2/[3\sqrt{2\log(2d)}]$. Then for any $\delta \in (0,1)$, with probability at least $1-\delta$ over the random training samples, the population risk satisfies*

$$\mathcal{R}(\widehat{\Theta}) \leq \frac{3\|f\|_{\mathcal{B}}^2}{LD} + \left(4\|f\|_{\mathcal{B}} + 1\right) \frac{3(4+\lambda)\sqrt{2\log(2d)} + 2}{\sqrt{n}} + 4\sqrt{\frac{2\log(14/\delta)}{n}}. \tag{3.26}$$

One unsatisfactory aspect of this result is that it is proved for a Barron function, not functions in the flow-induced space.

3.8. The loss function and the loss landscape

The a priori estimates for the regularized models establish the existence of accurate approximations to the target function in the hypothesis space. The next question is how to find them. At this point, there is a vast amount of experience suggesting that one can find accurate solutions using simple gradient-based algorithms, without any explicit regularization, but the result may depend sensitively on the choice of the hyperparameters, such as the network parameters, the initialization of the training algorithms, the learning rate, etc. Sensitive dependence on the network parameters suggests that the landscape of the loss function changes qualitatively as these parameters change.

At a first sight, it is quite surprising that simple gradient-based algorithms such as the gradient descent can work at all. After all, the loss function, say the empirical risk, is a nonconvex function of many variables with potentially very complicated landscape. In the case of molecular structural optimization such as protein folding, gradient descent would get stuck very quickly at a bad local minima. In the case of training neural network models, one can often avoid this by tuning the hyperparameters in the training algorithm. Obviously, this means that the landscape of the molecular structural optimization and the landscape for training neural network models are qualitatively very different. Therefore one first issue might be to understand how the landscape looks. In this direction, one important result is that of Cooper who considered overparametrized neural networks with a smooth activation

function and characterized the structure of the set of global minima [17]. Cooper proved that the locus of the global minima is generically (i.e., possibly after an arbitrarily small change to the data set) a smooth $(m-n)$-dimensional submanifold of $\mathbb{R}^m$ where m is the number of free parameters in the neural network model and n is the training data size.

3.9. Training dynamics

Two-layer neural networks with mean-field scaling. "Mean-field" is a notion in statistical physics that describes a particular form of the interaction between particles. In the mean-field situation, particles interact with each other only through a mean-field formed through the collective effort of all the particles. The most elegant mean-field picture in machine learning is found in the case of two-layer neural networks: If one views the neurons as interacting particles, then these particles only interact with each other through the function represented by the neural network, which is the mean-field in this case. This observation was first made in [15,75,84,92]. By taking the hydrodynamic limit for the gradient flow of finite neuron systems, these authors obtained a continuous integral differential equation that describes the evolution of the probability measure for the weights associated with the neurons.

Given the two-layer neural network model

$$f_m(\boldsymbol{x}) = \frac{1}{m}\sum_j a_j \sigma\left(\boldsymbol{w}_j^T \boldsymbol{x}\right),$$

let

$$I(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m) = \mathcal{R}(f_m), \quad \boldsymbol{u}_j = (a_j, \boldsymbol{w}_j).$$

Consider the gradient descent dynamics

$$\frac{d\boldsymbol{u}_j}{dt} = -m\nabla_{\boldsymbol{u}_j} I(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m), \quad \boldsymbol{u}_j(0) = \boldsymbol{u}_j^0, \quad j \in [m]. \tag{3.27}$$

Lemma 22. *Let*

$$\rho(d\boldsymbol{u}, t) = \frac{1}{m}\sum_j \delta_{\boldsymbol{u}_j(t)}.$$

Then the gradient descent dynamics (3.27) *can be expressed equivalently as*

$$\partial_t \rho = \nabla(\rho \nabla V), \quad V = \frac{\delta \mathcal{R}_n}{\delta \rho}. \tag{3.28}$$

Equation (3.28) is the mean-field equation that describes the evolution of the probability distribution for the weights associated with each neuron. The lemma above simply states that (3.28) is satisfied for the finite neuron system without the need to take the infinite particle limit.

It is well known that (3.28) is the gradient flow of $\mathcal{R}$ under the Wasserstein metric. This brings the hope that the mathematical tools developed in the theory of optimal transport can be brought to bear for the analysis of (3.28) [95]. In particular, we would like to use these tools to study the qualitative behavior of the solutions of (3.28) as $t \to \infty$. Unfortunately, straightforward application of the results from optimal transport theory requires that the risk functional be displacement convex [74], a property that rarely holds in ML. As a result, less than expected has been achieved using the optimal transport theory.

The one important result, due originally to Chizat and Bach [15], is the following. We will state the result for the population risk. Again we consider the ReLU activation function.

Theorem 23 ([15,16,99]). *Let $\{\rho_t\}$ be a solution of the Wasserstein gradient flow such that*

- *ρ_0 is a probability distribution on the cone $\Theta := \{|a|^2 \leq |\boldsymbol{w}|^2\}$.*
- *Every open cone in Θ has positive measure with respect to ρ_0.*

Then the following are equivalent:

- *The velocity potentials $\frac{\delta \mathcal{R}}{\delta \rho}(\rho_t, \cdot)$ converge to a unique limit as $t \to \infty$.*
- *$\mathcal{R}(\rho_t)$ decays to the global infimum value as $t \to \infty$.*

If either condition is met, the unique limit of $\mathcal{R}(\rho_t)$ is zero. If ρ_t also converges in the Wasserstein metric, then the limit ρ_∞ is a minimizer.

A few remarks are in order:

- There are further technical conditions for the theorem to hold.
- Convergence of subsequences of $\frac{\delta \mathcal{R}}{\delta \rho}(\rho_t, \cdot)$ is guaranteed by compactness.
- The first assumption on ρ_0 is a smoothness assumption needed for the existence of the gradient flow.
- The second assumption on ρ_0 is called *omnidirectionality*. It ensures that ρ can shift mass in any direction which reduces risk. The requirement that the support of the initial distribution be sufficiently large seems to be confirmed by practical experience.

Two-layer neural networks with conventional scaling. In practice, people often use the scaling (instead of the mean-field scaling)

$$f_m(\boldsymbol{x}; \boldsymbol{a}, \boldsymbol{W}) = \sum_{j=1}^{m} a_j \sigma\left(\boldsymbol{w}_j^T \boldsymbol{x}\right) = \boldsymbol{a}^T \sigma(\boldsymbol{W}\boldsymbol{x}).$$

A popular initialization [48,63] is as follows:

$$a_j(0) \sim \mathcal{N}(0, \beta^2), \quad \boldsymbol{w}_j(0) \sim \mathcal{N}(0, I/d),$$

where $\beta = 0$ or $1/\sqrt{m}$. We define the Gram matrix $K = (K_{ij}) \in \mathbb{R}^{n \times n}$ as

$$K_{i,j} = \frac{1}{n} \mathbb{E}_{\boldsymbol{w} \sim \pi_0}\left[\sigma(\boldsymbol{w}^T \boldsymbol{x}_i)\sigma(\boldsymbol{w}^T \boldsymbol{x}_j)\right].$$

In this case, a lot is known in the so-called *highly overparametrized regime*. In this part, for simplicity, we will assume that the domain of interest is the unit ball S^{d-1} instead of the unit cube.

There is both good and bad news. The good news is that one can prove exponential convergence to global minima of the empirical risk.

Theorem 24 ([21]). *Let $\lambda_n = \lambda_{\min}(K)$ and assume $\beta = 0$. For any $\delta \in (0,1)$, assume that $m \gtrsim n^2 \lambda_n^{-4} \delta^{-1} \ln(n^2 \delta^{-1})$. Then with probability at least $1 - 6\delta$, we have*

$$\mathcal{R}_n\big(\boldsymbol{a}(t), \boldsymbol{W}(t)\big) \le e^{-m\lambda_n t} \mathcal{R}_n\big(\boldsymbol{a}(0), \boldsymbol{W}(0)\big). \tag{3.29}$$

Now the bad news: the generalization property of the converged solution is no better than that of the associated random feature model, defined by freezing $\{\boldsymbol{w}_j\} = \{\boldsymbol{w}_j(0)\}$ and only training $\{a_i\}$.

The first piece of insight that the underlying dynamics in this regime is effectively linear is given in [19]. Jacot et al. [53] termed the effective kernel the "*neural tangent kernel*" and this terminology has got a lot of popularity. Later it was proved rigorously that in this regime, the entire gradient descent path for the two-layer neural network model is uniformly close to that of the associated random feature model [3,31].

Theorem 25 ([31]). *Let $\boldsymbol{W}_0 = \boldsymbol{W}(0)$. Denote by $f_m(\cdot; \tilde{\boldsymbol{a}}, \boldsymbol{W}_0)$ the solution of the gradient descent dynamics for the random feature model. Under the same setting as in Theorem 24, we have*

$$\sup_{\boldsymbol{x} \in S^{d-1}} \big| f_m\big(\boldsymbol{x}; \boldsymbol{a}(t), \boldsymbol{W}(t)\big) - f_m\big(\boldsymbol{x}; \tilde{\boldsymbol{a}}(t), \boldsymbol{W}_0\big)\big| \lesssim \frac{(1 + \sqrt{\ln(1/\delta)})^2 \lambda_n^{-1}}{\sqrt{m}}. \tag{3.30}$$

This can also be seen from the (m, n) hyperparameter space. Shown in Figure 3 are the heat maps of the test errors under the conventional and mean-field scaling, respectively. We see that the test error changes smoothly as m changes for the mean-field scaling. In contrast, there is a clear "phase transition" in the heat map for the conventional scaling where we see the coexistence of a good (darker region) phase with small test error and a bad (lighter

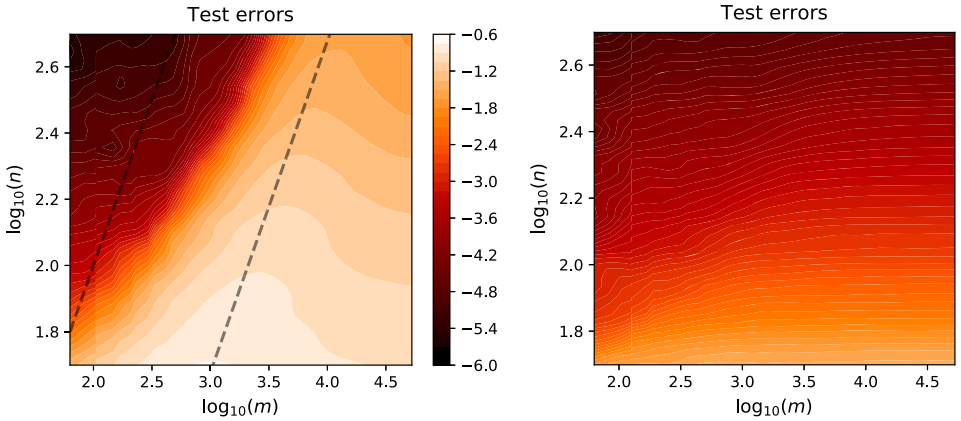

FIGURE 3

How the network width affects the test error of gradient descent solutions. The test errors are given in logarithmic scale. These experiments are conducted for the single-neuron target function with $d = 20$ and learning rate $\eta = 0.0005$. The two dashed lines correspond to $m = n/(d+1)$ (left) and $m = n$ (right), respectively. (Left) Conventional scaling. (Right) Mean-field scaling. For more details, see [72].

region) phase where the test error is much larger. This means that, in practice, one has to tune the network parameters so that they fall into the good phase. For the details of this study, we refer to [72].

From this, it is natural to speculate that the sensitive dependence of the performance on the hyperparameters is a consequence of this kinds of phase transition. For two layer neural networks, the phase diagram in the hyperparameter space is relatively simple. For more complicated neural network models, the phase diagram should be more complicated and tuning the parameters becomes a much harder task.

Hardness of training. In high dimensions, a "dynamic curse of dimensionality" may affect gradient descent training if the target function is not in Barron space.

Theorem 26 ([100]). *There exists f^* with Lipschitz constant and L^∞-norm bounded by* 1 *such that the parameter measures $\{\rho_t\}$ defined by the* 2*-Wasserstein gradient flow of either $\mathcal{R}_n$ or $\mathcal{R}$ satisfy*

$$\limsup_{t\to\infty}\left[t^\gamma\,\mathcal{R}(\rho_t)\right]=\infty$$

for all $\gamma > \frac{4}{d-2}$.

What makes matters worse is that even for functions in the Barron space, a dynamic CoD might also happen. Livni et al. [67] show that learning Barron functions is equivalent to solving some well-known hard problems in cryptography. This means that learning Barron functions is computationally as hard as breaking a cryptosystem. Such results are powerful but abstract. In the following, we provide an explicit understanding from the perspective of learning orthonormal classes, which is a reinterpretation of the results in [73, 87].

Consider a subset of the Barron space over $X=[0,1]^d$: $\mathcal{F}=\{f_{\boldsymbol{w}}=2\sin(2\pi \boldsymbol{w}^T\cdot):\sum_{i=1}^d w_i\le d, w_i\in\mathbb{N}_+\}$. Note that the following statements hold: (1) $|\mathcal{F}|\ge\exp(d)$; (2) $\langle f_{\boldsymbol{w}}, f_{\boldsymbol{w}'}\rangle=\delta_{\boldsymbol{w},\boldsymbol{w}'}$; (3) $\|f\|_{\mathcal{B}}\le Cd^2$, $\forall f\in\mathcal{F}$. Statements (1) and (2) are quite obvious and a proof can be found in [7]. Statement (3) directly follows from Theorem 3. Consider learning the function in $\mathcal{F}$ using the parametric model $h(\cdot;\theta)$ that includes, but is not limited to, the two-layer neural network model. Let $\mathcal{R}^f(\theta)=\mathbb{E}_{\boldsymbol{x}}[(h(\boldsymbol{x};\theta)-f(\boldsymbol{x}))^2]$. Notice that $\nabla_\theta\mathcal{R}^f(\theta)=2\mathbb{E}_{\boldsymbol{x}}[(h(\boldsymbol{x};\theta)-f(\boldsymbol{x}))\nabla_\theta h(\boldsymbol{x};\theta)]=C(\theta)-2\langle\nabla_\theta h(\cdot;\theta),f\rangle$. Let ν be the uniform distribution over $\mathcal{F}$. Then,

$$\begin{aligned}\operatorname{var}_{f\sim\nu}\left(\nabla\mathcal{R}^f(\theta)\right)&\le 4\mathbb{E}_{f\sim\nu}\left\langle\nabla_\theta h(\cdot;\theta),f\right\rangle^2=\frac{4}{|\mathcal{F}|}\sum_{f\in\mathcal{F}}\left\langle\nabla_\theta h(\cdot;\theta),f\right\rangle^2\\&\le\frac{4\mathbb{E}_{\boldsymbol{x}}\|\nabla_\theta h(\boldsymbol{x};\theta)\|^2}{|\mathcal{F}|},\end{aligned}\tag{3.31}$$

where the last inequality uses the fact that the functions in $\mathcal{F}$ are orthonormal.

Since $|\mathcal{F}|=\exp(d)$, the variance of the gradient with respect to different target function is exponentially small as d increases. This means that the gradient can barely distinguish different target functions. As a result, gradient-based optimizations algorithms are unlikely to succeed.

These hardness results suggest that the Barron space is likely to be too large for studying the training of two-layer neural networks. It is an important open problem to identify the right function space, in which the functions can be learned in polynomial time by two-layer neural networks.

3.10. Other results

Classification problems. A binary classification problem can be approached as a regression problem with the additional knowledge that the target function only takes the values ± 1. This a priori knowledge gives us different mathematical means, and we commonly choose to interpret $\{f > 0\}$ as $\{f \approx 1\}$ and similarly for negative values. It is therefore not necessary that f should take a particular value, but only that f have the correct sign (and possibly be bounded away from zero). This is encoded in the common hinge loss and logistic loss functions

$$\ell_{\text{hinge}}(h, y) = \max\{0, 1 - hy\}, \quad \ell_{\log}(h, y) = \frac{\log(1 + \exp(-hy))}{\log 2},$$

which primarily force alignment between the classifier h and the label $y \in \{-1, 1\}$. Both the hinge loss and the logistic loss differ from the ℓ^2-loss geometrically from the optimization perspective: While the ℓ^2-loss vanishes at exactly one point, hinge-loss vanishes whenever the classifier has the correct sign and magnitude ≥ 1, whereas the logistic loss never vanishes. The risk functional therefore has a much larger set of minimizers or none at all in typical classification problems.

Another key difference is the fact that we are minimizing a *surrogate loss*. While our goal is to minimize the measure of the misclassified set $\mathbb{E}_{(\boldsymbol{x},y)\sim\mu}[1_{\{f(\boldsymbol{x})\cdot y\leq 0\}}]$, we use convex loss functions ℓ which bound the zero–one loss function from above:

$$\ell(h, y) \geq 1_{\{h\cdot y\leq 0\}}.$$

The bounds on the true risk functional to minimize are therefore coarser by nature.

The nonexistence of minimizers for logistic loss has interesting implications from the optimization perspective. It was shown in [16] that as the risk decays to zero along a gradient flow trajectory, the geometry of a two-layer neural network adapts not only to correct classification, but also a higher-order optimality condition (maximum margin classification), where the "confidence" (or margin)

$$\min_{(\boldsymbol{x},y)\in \operatorname{spt}(\mu)} h(\boldsymbol{x}) \cdot y$$

becomes as large as possible. The notion of margin is easiest to interpret for linear classifiers, where it corresponds to the distance to the decision boundary, and harder to interpret in classes of nonlinear functions such as neural networks.

In multiclass classification, a similar philosophy holds, but the classifier has to align with the vectors $\boldsymbol{e}_1, \ldots, \boldsymbol{e}_k$ corresponding to the k classes, rather than the directions ± 1. The most popular loss functional in this case is the cross-entropy loss, which generalizes the logistic loss. Just as the logistic loss, the cross-entropy loss function does not admit minimizers either.

The frequency principle. One of the most interesting observations for training dynamics is the so-called "frequency principle": During training, the low frequency components in the target function tend to be recovered earlier than the high ones [103]. This is opposite to the situation we usually see in numerical analysis, for example, the numerical solution of elliptic PDEs. It is well known that when using iterative algorithms to solve the algebraic equations obtained from the numerical discretization of elliptic PDEs, it takes longer to remove the low frequency errors than the high frequency errors. This is why multi-grid methods are useful there. In ML, the opposite seems to be true.

The reason behind this is as follows. Roughly speaking, when solving PDEs, the high frequency components correspond roughly to large eigenvalues of the underlying algebraic system. As we have seen earlier, the training dynamics in machine learning is more like solving some integral equation, therefore the high frequency components correspond roughly to small eigenvalues. This should be an important avenue for understanding the training dynamics.

Generative models. Generative models are ways of approximating probability distributions using finite samples. One of the most well-known generative models is the generative adversarial network, or GAN. Given a sample set S, the empirical distribution formed from S, δ_S, can be considered as an approximation to the underlying probability distribution. This approximation is unsatisfactory since it cannot provide any new samples. However, it can be shown, or at least argued, that without any explicit regularization, generative models will always converge to the empirical distribution (see, for example, [39,104]). Therefore the merit of a generative model must be that during training, it can produce better approximations to the underlying probability distribution before ultimately converges to the empirical distribution. Theoretically, this means that one has to study the situation with "early stopping," not the ultimate convergence, and analyze whether the statement above really holds.

Results of this type have been proved for the so-called "bias potential" model [104]. However, it is fair to say that there are more open questions for the theoretical understanding of generative models than for the case of supervised learning. One of the difficulty is that generative models are not variational problems but rather game theory problems.

Machine learning of dynamical systems. Given a sample of time series, one would like to learn the underlying dynamical system that produced the time series. For these problems, besides CoD, there is also the issue of *curse of memory* [65]: The cost increases exponentially as memory increases. For linear dynamical systems, this issue has been analyzed quite thoroughly in [65].

Reinforcement learning. Reinforcement learning (RL) is an area of machine learning that is concerned with how an agent should interact with an unknown environment in order to maximize the expected cumulative reward [93]. To deal with practical problems that involve a large number of states or in high dimensions, one needs to introduce function approximation for the value or policy functions. Indeed, RL has had remarkable success in Atari games [76], Go [90], and robotics [61] using deep neural network approximations. Despite the

practical success of RL with function approximation, most existing theoretical results is only applicable to the tabular setting [4,5,55], in which both the state and action spaces are finite and no function approximation is involved. Relatively simple function approximation methods, such as the linear model [56,105], have been studied recently. RL with kernel function approximation has been studied in [37,68,69,106], and RL with neural network approximation has been studied in [36,69,97]. These results require the existence of a reference distribution such that all possible state–action distributions under the admissible policies are close to in a certain sense, and [68] shows the necessity of this type of assumptions.

In a way, the mathematical issues for RL are a lot like those for supervised learning, but more complicated. Specifically,

- For the approximation error: Analyze the conditions on the reward function and transition probability under which the value or policy functions can be efficiently approximated by neural networks. Since the value and policy functions are obtained from the Bellman equation, this question is related to the new regularity theory for PDEs discussed below.
- For the generalization error: This is the sample complexity problem. It is similar in spirit to the Rademacher complexity except that there is an additional dynamic component.
- For the optimization error: Again the dynamic component complicates things. This is dynamics within dynamics: The optimization algorithm involves dynamics. Within that there is the dynamics of the underlying problem such as the dynamics of the Go game.

New regularity theory of PDEs. The approximation theory discussed earlier suggests that in high dimension, the classical smoothness-based function spaces such as Sobolev spaces should be replaced by new spaces such as the Barron space or Barron's spectral space. It is natural to ask whether the solutions of prototypical PDEs lies in these new spaces or whether one can develop a regularity theory for the relevant PDEs in these new spaces. This issue is of practical significance because of the success of ML-based algorithms for solving PDEs in high dimension (see Section 2). In this direction, [101] considered the simplest PDEs such as the Poisson equation, heat equation, and Hamilton–Jacobi equation, and studied whether the solutions lie in Barron space if the data does. Lu et al. [71] carried out a more thorough analysis and developed a regularity theory for Barron's spectral space. Also of relevance is the work in [40,52].

4. MACHINE LEARNING FROM A CONTINUOUS VIEWPOINT

To define a machine learning model, one only needs two things: A way of representing functions and a way of finding the parameters in the representation. The latter is usually formulated as an optimization problem. Neural networks are special classes of functions. In contrast to piecewise polynomials, they can be defined without using a mesh. This "contin-

uous" nature is quite helpful when constructing numerical algorithms. One is naturally led to ask: Can we push this continuous nature further?

One interesting idea proposed in [23,41] is to use the solutions of ordinary differential equations (ODEs) to represent trial functions. This proposal is now made popular through the name of *neural ODEs* [14]. ResNets can be viewed as discretizations of neural ODEs, and the back-propagation algorithm can be viewed as a particular way of solving the adjoint equation for the gradients of the solutions [14,23].

A more systematic presentation of the continuous formulation is found in [32]. The framework suggested there consists of the following main components:

- representation of functions;
- the variational problem for minimizing the population risk;
- gradient flow for the variational problem.

Since we are aiming at high-dimensional problems, the function representation should be of a probabilistic nature. E et al. [32] suggested two classes of representations, integral transform-based and flow-based.

Once we have a continuous formulation, one can then discretize the continuous formulation to obtain concrete algorithms. Again, since we are aiming at high dimension, the particle method is the most natural algorithm for discretizing the dynamic equations. We will see that some of the most popular neural network-based ML algorithms can be derived this way. At the same time, one can also come up with new algorithms.

4.1. Integral transform-based representation

Consider the (parametric) representation

$$f(\boldsymbol{x};a,\pi)=\int_{\mathbb{R}^d} a(\boldsymbol{w})\sigma(\boldsymbol{w}^T\boldsymbol{x})\pi(d\boldsymbol{w})=\mathbb{E}_{\boldsymbol{w}\sim\pi}a(\boldsymbol{w})\sigma(\boldsymbol{w}^T\boldsymbol{x}), \tag{4.1}$$

or

$$f(\boldsymbol{x};\rho)=\mathbb{E}_{(a,\boldsymbol{w})\sim\rho}a\sigma(\boldsymbol{w}^T\boldsymbol{x}). \tag{4.2}$$

Given a target function f^*, the variational problem for minimizing the population risk is given by

$$\min_{\theta}\mathcal{R}(\theta),\quad \mathcal{R}(\theta)=\mathbb{E}_{\boldsymbol{x}\sim\mu}\big(f(\boldsymbol{x};\theta)-f^*(\boldsymbol{x})\big)^2,$$

where θ denotes abstract parameters. For (4.1), $\theta=(a,\pi)$; for (4.2), $\theta=\rho$. These two ingredients together form the variational problem we are interested in. We can either discretize this variational problem and then solve the resulted discretized optimization problem, or formulate an optimization problem at the continuous level and then discretize. We will discuss the latter approach.

To define the gradient flow, we borrow ideas from nonequilibrium statistical physics [51]. We regard the population risk as the free energy, and the parameter θ as the "order parameter". As in [51], we can divide the order parameters into two different classes, the

conserved and nonconserved ones. For example, as a probability measure, π and ρ are conserved order parameters. In contrast, the coefficient a is nonconserved.

For nonconserved order parameters, we use the so-called "model A" gradient flow, e.g.,

$$\frac{\partial a}{\partial t} = -\frac{\delta \mathcal{R}}{\delta a}.$$

For conserved order parameters, we use "model B" gradient flow, e.g.,

$$\frac{\partial \pi}{\partial t} + \nabla \cdot \mathbf{J} = 0,$$

where

$$\mathbf{J} = \pi \boldsymbol{v}, \quad \boldsymbol{v} = -\nabla V, \quad V = \frac{\delta \mathcal{R}}{\delta \pi}.$$

It is instructive to look at some specific examples. For the first example, we use the representation (4.1). We fix π and optimize over a. The gradient flow in this case is given by

$$\partial_t a(\boldsymbol{w}, t) = -\frac{\delta \mathcal{R}}{\delta a}(\boldsymbol{w}, t) = -\int a(\tilde{\boldsymbol{w}}, t) K(\boldsymbol{w}, \tilde{\boldsymbol{w}}) \pi(d\tilde{\boldsymbol{w}}) + \tilde{f}(\boldsymbol{w}), \tag{4.3}$$

where

$$K(\boldsymbol{w}, \tilde{\boldsymbol{w}}) = \mathbb{E}_{\boldsymbol{x}}\big[\sigma(\boldsymbol{w}^T \boldsymbol{x})\sigma(\tilde{\boldsymbol{w}}^T \boldsymbol{x})\big], \quad \tilde{f}(\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{x}}\big[f^*(\boldsymbol{x})\sigma(\boldsymbol{w}^T \boldsymbol{x})\big].$$

This is an integral equation with a symmetric positive definite kernel.

As an example of the conservative gradient flow, let us consider the representation

$$f(\boldsymbol{x}) = \mathbb{E}_{\boldsymbol{u}\sim\rho}\phi(\boldsymbol{x}, \boldsymbol{u}),$$

where ϕ is some general feature function. One example is $\boldsymbol{u} = (a, \boldsymbol{w})$, $\phi(\boldsymbol{x}, \boldsymbol{u}) = a\sigma(\boldsymbol{w}^T \boldsymbol{x})$. Let

$$V(\boldsymbol{u}) = \frac{\delta \mathcal{R}}{\delta \rho}(\boldsymbol{u}) = \mathbb{E}_{\boldsymbol{x}}\big[\big(f(\boldsymbol{x}) - f^*(\boldsymbol{x})\big)\phi(\boldsymbol{x}, \boldsymbol{u})\big] = \int \tilde{K}(\boldsymbol{u}, \tilde{\boldsymbol{u}})\rho(d\tilde{\boldsymbol{u}}) - \tilde{f}(\boldsymbol{u})$$

be the potential with a kernel $\tilde{K}$ defined similarly to K, then the model B gradient flow dynamics is given by

$$\partial_t \rho = \nabla(\rho \nabla V). \tag{4.4}$$

This is the same as the mean-field equation derived in [75, 84, 92]. For an interesting modification of this model to improve convergence, we refer to [83].

Next, we turn to the discretization of the continuous formulation. There are several levels of discretization to consider:

- Discretizing the variational problem. In the current setting, this is straightforward: By using data, we discretize the population risk into the empirical risk.
- Discretizing the function representation and the gradient flow. This can be done using a number of different numerical methods. In low dimensions, the spectral method can be a powerful tool. In high dimensions, the most obvious choice is the particle method since this is the dynamic version of Monte Carlo. One can

also use the smoothed particle method which has shown better performance at least on low dimensional problems.

As an example, consider (4.1). If we use

$$\pi(d\boldsymbol{w}) \sim \frac{1}{m}\sum_j \delta_{\boldsymbol{w}_j}, \quad a(\boldsymbol{w}_j, t) \sim a_j(t)$$

to discretize (4.3), we obtain

$$\frac{d}{dt}a_j(t) = -\frac{1}{m}\sum_k K(\boldsymbol{w}_j, \boldsymbol{w}_k)a_k(t) + \tilde{f}(\boldsymbol{w}_j).$$

This is exactly the gradient descent dynamics for the random feature model.

Next, consider the integral differential equation (4.4). If we use the particle method discretization, we have

$$\rho(da, d\boldsymbol{w}, t) \sim \frac{1}{m}\sum_j \delta_{(a_j(t),\boldsymbol{w}_j(t))} = \frac{1}{m}\sum_j \delta_{\boldsymbol{u}_j(t)}$$

and the discretized problem becomes

$$\frac{d\boldsymbol{u}_j}{dt} = -\nabla_{\boldsymbol{u}_j} I(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m),$$

where

$$I(\boldsymbol{u}_1, \ldots, \boldsymbol{u}_m) = \mathcal{R}(f_m), \quad \boldsymbol{u}_j = (a_j, \boldsymbol{w}_j), \quad f_m(\boldsymbol{x}) = \frac{1}{m}\sum_j a_j \sigma\left(\boldsymbol{w}_j^T \boldsymbol{x}\right).$$

This is exactly the gradient descent dynamics for two-layer neural networks under the mean-field scaling.

4.2. Flow-based representation

Consider the following flow-based representation:

$$\begin{aligned} \boldsymbol{z}_0^{\boldsymbol{x}} &= V\boldsymbol{x}, \\ \frac{d\boldsymbol{z}_\tau^{\boldsymbol{x}}}{d\tau} &= \mathbb{E}_{\boldsymbol{w}\sim\rho_\tau}\boldsymbol{\phi}(\boldsymbol{z}, \boldsymbol{w}), \quad \forall \tau \in [0,1], \\ f(\boldsymbol{x};\theta) &= \mathbf{1}^T \boldsymbol{z}_1^{\boldsymbol{x}}, \end{aligned} \tag{4.5}$$

where $\boldsymbol{w} \in \Omega$ and $\boldsymbol{\phi} : \mathbb{R}^D \times \Omega \mapsto \mathbb{R}^D$, V is a $D \times d$ matrix. For simplicity, we will fix V. The parameters are then $\theta = \rho = (\rho_\tau)_{\tau\in[0,1]}$, a sequence of probability measures.

The form of the right-hand side in (4.5) is chosen because of the following two considerations. The first is that it is an integral transformation-based representation that we just discussed. More importantly, it arises as the natural continuum limit of a suitable ResNet model with random parameters [33].

Minimizing the population risk using this flow-based representation is then a control problem where the population risk serves as the objective function, and the parameters $\rho = (\rho_\tau)_{\tau\in[0,1]}$ serve as the control. One useful tool from the control perspective is Pontryagin's maximum principle. To state this maximum principle, denote by $\Sigma := \{\rho : [0,1] \mapsto \mathcal{P}_2(\Omega)\}$,

the space of all feasible controls, and define the Hamiltonian $H : \mathbb{R}^d \times \mathbb{R}^d \times \mathcal{P}_2(\Omega) :\mapsto \mathbb{R}$ as

$$H(\boldsymbol{z}, \boldsymbol{p}, \mu) = \mathbb{E}_{\boldsymbol{u}\sim\mu}\big[\boldsymbol{p}^T \boldsymbol{\phi}(\boldsymbol{z}, \boldsymbol{u})\big].$$

Here $\boldsymbol{z}, \boldsymbol{p} \in \mathbb{R}^D$ and $\mu \in \mathcal{P}_2(\Omega)$. $\boldsymbol{p}$ is the costate that corresponds to the state $\boldsymbol{z}$.

Pontryagin's maximum principle states that the solutions of this control problem must satisfy

$$\rho_\tau^* = \text{argmax}_\rho \, \mathbb{E}_{\boldsymbol{x}}\big[H\big(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{p}_\tau^{\boldsymbol{x}}, \rho\big)\big], \quad \forall \tau \in [0, 1], \tag{4.6}$$

and for each $\boldsymbol{x}$, $(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{p}_\tau^{\boldsymbol{x}})$ are defined by the forward/backward equations:

$$\begin{aligned} \frac{d\boldsymbol{z}_\tau^{\boldsymbol{x}}}{d\tau} &= \nabla_{\boldsymbol{p}} H = \mathbb{E}_{\boldsymbol{u}\sim\rho_\tau^*}\big[\boldsymbol{\phi}\big(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{u}\big)\big], \\ \frac{d\boldsymbol{p}_\tau^{\boldsymbol{x}}}{d\tau} &= -\nabla_{\boldsymbol{z}} H = \mathbb{E}_{\boldsymbol{u}\sim\rho_\tau^*}\big[\nabla_{\boldsymbol{z}}^T \boldsymbol{\phi}\big(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{u}\big)\boldsymbol{p}_\tau^{\boldsymbol{x}}\big], \end{aligned} \tag{4.7}$$

with the boundary conditions:

$$\boldsymbol{z}_0^{\boldsymbol{x}} = \boldsymbol{x}, \tag{4.8}$$

$$\boldsymbol{p}_1^{\boldsymbol{x}} = 2f\big(\boldsymbol{x}; \rho_1^* - f^*(\boldsymbol{x})\big)\mathbf{1}. \tag{4.9}$$

With this, one can then construct maximum principle-based algorithms. This was first done in [64] and it was based on an extension of the *method of successive approximation (MSA)*. This is an iterative algorithm that alternates between solving the Hamiltonian system for the states and costates and finding the optimal parameters at each step. Symbolically, one can write it as, at the step k:

- Solve
$$\frac{d\boldsymbol{z}_\tau^k}{d\tau} = \nabla_{\boldsymbol{p}} H\big(\boldsymbol{z}_\tau^k, \boldsymbol{p}_\tau^k, \theta_\tau^k\big), \quad \boldsymbol{z}_0^k = V\boldsymbol{x}.$$
- Solve
$$\frac{d\boldsymbol{p}_\tau^k}{d\tau} = -\nabla_{\boldsymbol{z}} H\big(\boldsymbol{z}_\tau^k, \boldsymbol{p}_\tau^k, \theta_\tau^k\big), \quad \boldsymbol{p}_1^k = 2\big(f(\boldsymbol{x}; \theta^k) - f^*(\boldsymbol{x})\big)\mathbf{1}.$$
- Set $\theta_\tau^{k+1} = \text{argmax}_{\theta\in\Theta} H(\boldsymbol{z}_\tau^k, \boldsymbol{p}_\tau^k, \theta)$, for each $\tau \in [0, 1]$.

Compared with the usual gradient descent-based algorithms, the advantage is that the optimization problems are decoupled for different values of τ. Li et al. [64] presented numerical evidence which suggests that an extended version of this algorithm is quite competitive, compared with several different versions of SGD.

The gradient flow for this model was derived in [32]. For any $\rho^1, \rho^2 \in X$, consider the following metric:

$$\mathcal{D}^2(\rho^1, \rho^2) := \int_0^1 W_2^2\big(\rho_\tau^1, \rho_\tau^2\big) d\tau,$$

where $W_2(\cdot, \cdot)$ is the 2-Wasserstein distance.

Proposition 27. *The gradient flow in the metric space* $(\Sigma, \mathcal{D})$ *for the population risk is given by*

$$\partial_t \rho_\tau(\boldsymbol{w}, t) = \nabla_{\boldsymbol{w}} \cdot \left(\rho_\tau \mathbb{E}_{\boldsymbol{x}}\left[\boldsymbol{v}\left(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{p}_\tau^{\boldsymbol{x}}, \boldsymbol{w}\right)\right]\right), \quad \forall \tau \in [0,1], \tag{4.10}$$

where

$$\boldsymbol{v}(\boldsymbol{z}, \boldsymbol{p}, \boldsymbol{w}) = \nabla_{\boldsymbol{w}} \frac{\delta H}{\delta \mu} = \nabla_{\boldsymbol{w}}^T \boldsymbol{\phi}(\boldsymbol{z}, \boldsymbol{w}) \boldsymbol{p},$$

and for each $\boldsymbol{x}$, $(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{p}_\tau^{\boldsymbol{x}})$ *satisfies* (4.7) *and* (4.8) *with* ρ^* *replaced by* ρ *at time* t.

This is a one parameter family of coupled flows. For this flow, the energy dissipation relation is given by

$$\frac{d\mathcal{R}}{dt} = -\int_0^1 \mathbb{E}_{\boldsymbol{w} \sim \rho_\tau(\cdot;t)}\left[\left\| \mathbb{E}_{\boldsymbol{x}} \nabla_{\boldsymbol{w}}^T \boldsymbol{\phi}\left(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{w}\right) \boldsymbol{p}_\tau^{\boldsymbol{x}} \right\|^2\right] d\tau. \tag{4.11}$$

Consider now the discretization of the flow using the particle method. Letting $\rho_\tau(\cdot, t) = \frac{1}{m}\sum_{j=1}^m \delta(\boldsymbol{w}_\tau^j(t) - \cdot)$, the discretized gradient flow is given by

$$\begin{aligned}
\frac{d\boldsymbol{z}_\tau^{\boldsymbol{x}}}{d\tau} &= \frac{1}{m}\sum_{j=1}^m \boldsymbol{\phi}\left(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{w}_\tau^j\right), \quad \tau \in [0,1], \\
\frac{d\boldsymbol{p}_\tau^{\boldsymbol{x}}}{d\tau} &= -\frac{1}{m}\sum_{j=1}^m \nabla_{\boldsymbol{z}} \boldsymbol{\phi}\left(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{w}_\tau^j\right) \boldsymbol{p}_\tau^{\boldsymbol{x}}, \quad \tau \in [0,1], \\
\frac{d\boldsymbol{w}_\tau^j}{dt} &= -\mathbb{E}_{\boldsymbol{x}}\left[\nabla_{\boldsymbol{w}}^T \boldsymbol{\phi}\left(\boldsymbol{z}_\tau^{\boldsymbol{x}}, \boldsymbol{w}_\tau^j\right)^T \boldsymbol{p}_\tau^{\boldsymbol{x}}\right], \quad j = 1, \ldots, m.
\end{aligned} \tag{4.12}$$

Upon further discretizing the flow-based representation, one essentially recovers the gradient descent algorithm for ResNets together with back-propagation (for more details, see [32]).

The gradient descent-based algorithm and the maximum principle-based algorithms are two representative classes of training algorithms for deep neural networks. There are two major components in these algorithms: the propagation and back-propagation of the states and the costates, and the optimization of the parameters. In gradient-descent algorithms, for each iteration of the gradient descent, one performs a full cycle of forward and backward propagation. In maximum principle-based algorithms, for each cycle of the forward and backward propagation, one performs the full optimization. These two classes of algorithms stand at the opposite extreme as far as the balance of these two components are concerned. Obviously, the most efficient algorithm should lie somewhere in-between.

Another interesting question is the comparison between the mean field and the continuous philosophies. In the simplest setting, the mean-field and the continuous formulation give rise to the same continuous model. However, one should note that their starting point is quite different:

- For the mean field approach: discrete → continuous by taking the hydrodynamic limit, as in the study of interacting particle systems in statistical physics.
- For the continuous formulation: continuous → discrete by discretization. This viewpoint is more like the one in classical numerical analysis where one starts from continuous problems and then discretize.

Continuous formulation allows us to think about machine learning "outside the box" of neural network models. It also seems to be a "double-sided sword": While the continuous formulation seems to be quite attractive, it is difficult to initialize. In practice, it seems that initializing using i.i.d. random samples under the conventional scaling leads to better performance. This is still a puzzle that needs to be resolved.

5. SOME PERSPECTIVES AND CONCLUDING REMARKS

What have we really learned? Perhaps the single most important thing is that neural network-based machine learning is a very powerful tool for overcoming the CoD, or for discrete problems, the *combinatorial explosion*. This should be the most important guiding principle for designing new algorithms, trying new applications, or developing the theory. Neural networks might also be useful for problems with very few degrees of freedom, but at the moment, we still lack convincing evidence in this regard.

One of the most exciting recent development is the application of machine learning to science. AlphaFold2 and DeePMP are two of the most representative examples. The former is a powerful solution of a fundamental problem in science using data-driven methods. The latter is a powerful extension of a classical theoretical tool, namely molecular dynamics, that substantially advanced its realm of applicability. As we discussed in Section 2, machine learning seems to provide the missing tool for realizing the goals put forward in the multiscale modeling program. In addition, using machine learning to improve the efficiency of experimental work is also an area with a lot of promise. Indeed, one can argue that *AI for Science* has been the most exciting development in AI or science during the last couple of years, and it is changing the paradigm with which we do science.

On the theoretical side, even though we are still quite far from having a satisfactory theory for neural network-based machine learning, the roadmap to such a theory is emerging. This roadmap includes understanding the approximation theory, generalization gap, the landscape for the training problem, dynamical path during training, the difference between the landscape for the empirical and population risks, and so on. Perhaps more importantly, a consensus is starting to emerge regarding what the right questions are. One such consensus is that what is important is not the specific values of the neural network parameters, but rather their probability distribution. This underlies most of the theoretical advances discussed here.

Besides these abstract studies, there is also the need to study in more detail the structure of practical datasets. The fact that one can perform classification of images using neural network models suggests that the task itself is not so complicated, at least when represented using multilayer neural networks. It is worthwhile to look into the details of the structures of such a representation.

In addition to supervised learning, there is also the need to build some theoretical understanding of unsupervised learning, learning dynamical systems, reinforcement learning, as well as the new tasks that have emerged in the application of machine learning to scientific computing. The efforts to develop such an understanding is likely going to lead us to a new subject in mathematics, namely high-dimensional analysis.

Regarding the impact that machine learning will have on applied mathematics as a whole, we refer the readers to the article [24].

Finally, machine learning is not the ultimate solution of AI. It has a lot of problems, including the difficulties with interpretability, the need for a large training dataset, the vulnerability to adversarial attacks, and so on. Traditional rule-based methods are much better on these issues. Naturally one should ask whether it is possible to combine rule-based and learning-based approaches to build better AI algorithms.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my collaborators, particularly Roberto Car, Jiequn Han, Qianxiao Li, Jihao Long, Chao Ma, Cheng Tai, Han Wang, Stephan Wojtowytsch, Lei Wu, Hongkang Yang, and Linfeng Zhang, for their work and for the discussions we have had that helped to shape this report.

There is a vast amount of literature in machine learning and it grows very fast. We feel that we have included the most important literature that is relevant to the topics discussed here, but we apologize for any of the missing items. For additional literature, the reader is referred to the review articles [27,34].

FUNDING

This work was partially supported by a gift to the Princeton University from iFlytek. Part of the work is also supported by an ONR grant N00014-13-1-0338.

REFERENCES

[1] E. Abbe and C. Sandon, Poly-time universality and limitations of deep learning. 2020, arXiv:2001.02992.

[2] N. Aronszajn, Theory of reproducing kernels. *Trans. Amer. Math. Soc.* **68** (1950), no. 3, 337–404.

[3] S. Arora, S. S. Du, W. Hu, Z. Li, R. Salakhutdinov, and R. Wang, On exact computation with an infinitely wide neural net. 2019, arXiv:1904.11955.

[4] M. G. Azar, R. Munos, and H. Kappen, On the sample complexity of reinforcement learning with a generative model. In *International conference on machine learning*, 2012.

[5] M. G. Azar, I. Osband, and R. Munos, Minimax regret bounds for reinforcement learning. In *International conference on machine learning*, pp. 263–272, PMLR, 2017.

[6] F. Bach, Breaking the curse of dimensionality with convex neural networks. *J. Mach. Learn. Res.* **18** (2017), no. 19, 1–53.

[7] A. R. Barron, Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inf. Theory* **39** (1993), no. 3, 930–945.

[8] A. R. Barron, Approximation and estimation bounds for artificial neural networks. *Mach. Learn.* **14** (1994), no. 1, 115–133.

[9] J. Behler and M. Parrinello, Generalized neural-network representation of high-dimensional potential-energy surfaces. *Phys. Rev. Lett.* **98** (2007), no. 14, 146401.

[10] L. Bonati and M. Parrinello, Silicon liquid structure and crystal nucleation from ab initio deep metadynamics. *Phys. Rev. Lett.* **121** (2018), no. 26, 265701.

[11] L. Breiman, Hinging hyperplanes for regression, classification, and function approximation. *IEEE Trans. Inf. Theory* **39** (1993), no. 3, 999–1013.

[12] R. Car and M. Parrinello, Unified approach for molecular dynamics and density-functional theory. *Phys. Rev. Lett.* **55** (1985), no. 22, 2471.

[13] G. Carleo and M. Troyer, Solving the quantum many-body problem with artificial neural networks. *Science* **355** (2017), no. 6325, 602–606.

[14] R. T. Q. Chen, Y. Rubanova, J. Bettencourt, and D. K. Duvenaud, Neural ordinary differential equations. In *Advances in neural information processing systems*, pp. 6571–6583, 2018.

[15] L. Chizat and F. Bach, On the global convergence of gradient descent for over-parameterized models using optimal transport. In *Advances in neural information processing systems*, pp. 3036–3046, 2018.

[16] L. Chizat and F. Bach, Implicit bias of gradient descent for wide two-layer neural networks trained with the logistic loss. In *Conference on learning theory*, pp. 1305–1338, PMLR, 2020.

[17] Y. Cooper, The loss landscape of overparameterized neural networks. 2018, arXiv:1804.10200.

[18] G. Cybenko, Approximation by superpositions of a sigmoidal function. *Math. Control Signals Systems* **2** (1989), no. 4, 303–314.

[19] A. Daniely, SGD learns the conjugate kernel class of the network. In *Advances in neural information processing systems*, pp. 2422–2430, 2017.

[20] J. Dick, F. Y. Kuo, and I. H. Sloan, High-dimensional integration: the quasi-Monte Carlo way. *Acta Numer.* **22** (2013), 133–288.

[21] S. S. Du, X. Zhai, B. Poczos, and A. Singh, Gradient descent provably optimizes over-parameterized neural networks. In *International conference on learning representations*, 2019.

[22] W. E, *Principles of multiscale modeling*. Cambridge University Press, 2011.

[23] W. E, A proposal on machine learning via dynamical systems. *Commun. Math. Stat.* **5** (2017), no. 1, 1–11.

[24] W. E, The dawning of a new era in applied mathematics. *Not. Amer. Math. Soc.* **68** (2021), no. 4, 565–571.

[25] W. E and B. Engquist, The heterogenous multiscale methods. *Commun. Math. Sci.* **1** (2003), no. 1, 87–132.

[26] W. E, J. Han, and A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations. *Commun. Math. Stat.* **5** (2017), no. 4, 349–380.

[27] W. E, J. Han, and A. Jentzen, Algorithms for solving high dimensional PDEs: from nonlinear Monte Carlo to machine learning. 2020, arXiv:2008.13333.

[28] W. E, J. Han, and L. Zhang, Machine-learning-assisted modeling. *Physics Today* **74** (2021), no. 7, 36–41.

[29] W. E, C. Ma, and Q. Wang, Rademacher complexity and the generalization error of residual networks. *Commun. Math. Sci.* **18** (2020), no. 6, 1755–1774.

[30] W. E, C. Ma, and L. Wu, A priori estimates of the population risk for two-layer neural networks. *Commun. Math. Sci.* **17** (2019), no. 5, 1407–1425. 2018, arXiv:1810.06397.

[31] W. E, C. Ma, and L. Wu, A comparative analysis of the optimization and generalization property of two-layer neural network and random feature models under gradient descent dynamics. *Sci. China Math.* (2020), 1–24. 2019, arXiv:1904.04326.

[32] W. E, C. Ma, and L. Wu, Machine learning from a continuous viewpoint, I. *Sci. China Math.* **63** (2020), no. 11, 2233–2266.

[33] W. E, C. Ma, and L. Wu, The Barron space and the flow-induced function spaces for neural network models. *Constr. Approx.* (2021), 1–38.

[34] W. E, C. Ma, L. Wu, and S. Wojtowytsch, Towards a mathematical understanding of neural network-based machine learning: what we know and what we don't. *CSIAM Trans. Appl. Math.* **1** (2020), no. 4, 561–615.

[35] W. E and S. Wojtowytsch, Representation formulas and pointwise properties for Barron functions. 2020, arXiv:2006.05982.

[36] J. Fan, Z. Wang, Y. Xie, and Z. Yang, A theoretical analysis of deep Q-learning. In *Learning for dynamics and control*, pp. 486–489, PMLR, 2020.

[37] A.-m. Farahmand, M. Ghavamzadeh, C. Szepesvári, and S. Mannor, Regularized policy iteration with nonparametric function spaces. *J. Mach. Learn. Res.* **17** (2016), no. 1, 4809–4874.

[38] T. E. Gartner, L. Zhang, P. M. Piaggi, R. Car, A. Z. Panagiotopoulos, and P. G. Debenedetti, Signatures of a liquid–liquid transition in an ab initio deep neural network model for water. *Proc. Natl. Acad. Sci.* **117** (2020), no. 42, 26040–26046.

[39] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial nets. In *Advances in neural information processing systems*, pp. 2672–2680, 2014.

[40] P. Grohs, F. Hornung, A. Jentzen, and P. Von Wurstemberger, A proof that artificial neural networks overcome the curse of dimensionality in the numerical approximation of Black–Scholes partial differential equations. 2018, arXiv:1809.02362.

[41] E. Haber and L. Ruthotto, Stable architectures for deep neural networks. *Inverse Probl.* **34** (2017), no. 1, 014004.

[42] I. Han, J. T. McKeown, L. Tang, C.-Z. Wang, H. Parsamehr, Z. Xi, Y.-R. Lu, M. J. Kramer, and A. J. Shahani, Dynamic observation of dendritic quasicrystal growth upon laser-induced solid-state transformation. *Phys. Rev. Lett.* **125** (2020), no. 19, 195503.

[43] J. Han and W. E, Deep learning approximation for stochastic control problems. In *NIPS2016, Deep Reinforcement Learning Workshop*, 2016.

[44] J. Han, A. Jentzen, and W. E, Solving high-dimensional partial differential equations using deep learning. *Proc. Natl. Acad. Sci.* **115** (2018), no. 34, 8505–8510.

[45] J. Han, L. Zhang, R. Car, and W. E, Deep potential: a general representation of a many-body potential energy surface. *Commun. Comput. Phys.* **23** (2018), no. 3, 629–639.

[46] J. Han, L. Zhang, and W. E, Solving many-electron Schrödinger equation using deep neural networks. *J. Comput. Phys.* **399** (2019), 108929.

[47] B. Hanin and D. Rolnick, How to start training: the effect of initialization and architecture. In *Advances in neural information processing systems*, pp. 571–581, 2018.

[48] K. He, X. Zhang, S. Ren, and J. Sun, Delving deep into rectifiers: surpassing human-level performance on ImageNet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 1026–1034, 2015.

[49] J. Hermann, Z. Schätzle, and F. Noé, Deep-neural-network solution of the electronic Schrödinger equation. *Nat. Chem.* **12** (2020), no. 10, 891–897.

[50] S. Hochreiter, The vanishing gradient problem during learning recurrent neural nets and problem solutions. *Internat. J. Uncertain. Fuzziness Knowledge-Based Systems* **6** (1998), no. 02, 107–116.

[51] P. C. Hohenberg and B. I. Halperin, Theory of dynamic critical phenomena. *Rev. Modern Phys.* **49** (1977), no. 3, 435.

[52] M. Hutzenthaler, A. Jentzen, T. Kruse, T. Anh Nguyen, and P. von Wurstemberger, Overcoming the curse of dimensionality in the numerical approximation of semilinear parabolic partial differential equations. *Proc. R. Soc. A* **476** (2020), no. 2244, 20190630.

[53] A. Jacot, F. Gabriel, and C. Hongler, Neural tangent kernel: convergence and generalization in neural networks. In *Advances in neural information processing systems*, pp. 8580–8589, 2018.

[54] W. Jia, H. Wang, M. Chen, D. Lu, L. Lin, R. Car, W. E, and L. Zhang, Pushing the limit of molecular dynamics with ab initio accuracy to 100 million atoms with machine learning. In *Proceedings of the international conference for high performance computing, networking, storage and analysis, SC'20*, pp. 1–14, IEEE Press, 2020.

[55] C. Jin, Z. Allen-Zhu, S. Bubeck, and M. I. Jordan, Is Q-learning provably efficient? In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, pp. 4868–4878, 2018.

[56] C. Jin, Z. Yang, Z. Wang, and M. I. Jordan, Provably efficient reinforcement learning with linear function approximation. In *Conference on learning theory*, pp. 2137–2143, PMLR, 2020.

[57] L. K. Jones, A simple lemma on greedy approximation in Hilbert space and convergence rates for projection pursuit regression and neural network training. *Ann. Statist.* (1992), 608–613.

[58] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko, et al., Highly accurate protein structure prediction with AlphaFold. *Nature* **596** (2021), no. 7873, 583–589.

[59] Y. Khoo, J. Lu, and L. Ying, Solving parametric PDE problems with artificial neural networks. *European J. Appl. Math.* **32** (2021), no. 3, 421–435.

[60] J. M. Klusowski and A. R. Barron, Risk bounds for high-dimensional ridge function combinations including neural networks. 2016, arXiv:1607.01434.

[61] J. Kober, J. A. Bagnell, and J. Peters, Reinforcement learning in robotics: a survey. *Internat. J. Robot. Res.* **32** (2013), no. 11, 1238–1274.

[62] Y. LeCun, Y. Bengio, and G. Hinton, Deep learning. *Nature* **521** (2015), no. 7553, 436–444.

[63] Y. LeCun, L. Bottou, G. B. Orr, and K.-R. Müller, Efficient backprop. In *Neural networks: tricks of the trade*, pp. 9–48, Springer, 2012.

[64] Q. Li, L. Chen, C. Tai, and W. E, Maximum principle based algorithms for deep learning. *J. Mach. Learn. Res.* **18** (2017), no. 1, 5998–6026.

[65] Z. Li, J. Han, W. E, and Q. Li, On the curse of memory in recurrent neural networks: approximation and optimization analysis. In *International conference on learning representations*, 2020.

[66] Z. Li, N. Kovachki, K. Azizzadenesheli, B. Liu, K. Bhattacharya, A. Stuart, and A. Anandkumar, Fourier neural operator for parametric partial differential equations. 2020, arXiv:2010.08895.

[67] R. Livni, S. Shalev-Shwartz, and O. Shamir, On the computational efficiency of training neural networks. In *Advances in neural information processing systems*, pp. 855–863, 2014.

[68] J. Long and J. Han, Perturbational complexity by distribution mismatch: a systematic analysis of reinforcement learning in reproducing kernel Hilbert space. 2021, arXiv:2111.03469.

[69] J. Long, J. Han, and W. E, An L^2 analysis of reinforcement learning in high dimensions with kernel and neural network approximation. 2021, arXiv:2104.07794.

[70] J. Lu, Z. Shen, H. Yang, and S. Zhang, Deep network approximation for smooth functions. *SIAM J. Math. Anal.* **53** (2021), no. 5, 5465–5506.

[71] Y. Lu, J. Lu, and M. Wang, A priori generalization analysis of the deep Ritz method for solving high dimensional elliptic partial differential equations. In *Conference on learning theory*, pp. 3196–3241, PMLR, 2021.

[72] C. Ma, L. Wu, and W. E, The quenching-activation behavior of the gradient descent dynamics for two-layer neural network models. 2020, arXiv:2006.14450.

[73] E. Malach and S. Shalev-Shwartz, When hardness of approximation meets hardness of learning. 2020, arXiv preprint arXiv:2008.08059.

[74] R. J. McCann, A convexity principle for interacting gases. *Adv. Math.* **128** (1997), no. 1, 153–179.

[75] S. Mei, A. Montanari, and P.-M. Nguyen, A mean field view of the landscape of two-layer neural networks. *Proc. Natl. Acad. Sci.* **115** (2018), no. 33, E7665–E7671.

[76] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, Playing Atari with deep reinforcement learning. 2013, arXiv:1312.5602.

[77] T. Nakamura-Zimmerer, Q. Gong, and W. Kang, Adaptive deep learning for high-dimensional Hamilton–Jacobi–Bellman equations. *SIAM J. Sci. Comput.* **43** (2021), no. 2, A1221–A1247.

[78] E. Pardoux and S. Peng, Adapted solution of a backward stochastic differential equation. *Systems Control Lett.* **14** (1990), no. 1, 55–61.

[79] E. Pardoux and S. Peng, Backward stochastic differential equations and quasilinear parabolic partial differential equations. In *Stochastic partial differential equations and their applications*, pp. 200–217, Springer, 1992.

[80] D. Pfau, J. S. Spencer, A. G. Matthews, and W. M. C. Foulkes, Ab initio solution of the many-electron Schrödinger equation with deep neural networks. *Phys. Rev. Res.* **2** (2020), no. 3, 033429.

[81] A. Rahimi and B. Recht, Random features for large-scale kernel machines. In *Proceedings of the 20th international conference on neural information processing systems*, pp. 1177–1184, 2007.

[82] 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. *J. Comput. Phys.* **378** (2019), 686–707.

[83] G. Rotskoff, S. Jelassi, J. Bruna, and E. Vanden-Eijnden, Neuron birth-death dynamics accelerates gradient descent and converges asymptotically. In *International conference on machine learning*, pp. 5508–5517, 2019.

[84] G. Rotskoff and E. Vanden-Eijnden, Parameters as interacting particles: long time convergence and asymptotic error scaling of neural networks. In *Advances in neural information processing systems*, pp. 7146–7155, 2018.

[85] J. Schmidhuber, Deep learning in neural networks: an overview. *Neural Netw.* **61** (2015), 85–117.

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

[87] O. Shamir, Distribution-specific hardness of learning neural networks. *J. Mach. Learn. Res.* **19** (2018), no. 1, 1135–1163.

[88] J. W. Siegel and J. Xu, Approximation rates for neural networks with general activation functions. *Neural Netw.* **128** (2020), 313–321.

[89] J. W. Siegel and J. Xu, Characterization of the variation spaces corresponding to shallow neural networks. 2021, arXiv:2106.15002.

[90] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap, M. Leach, K. Kavukcuoglu, and D. Hassabis, Mastering the game of Go with deep neural networks and tree search. *Nature* **529** (2016), no. 7587, 484–489.

[91] J. Sirignano and K. Spiliopoulos, DGM: a deep learning algorithm for solving partial differential equations. *J. Comput. Phys.* **375** (2018), 1339–1364.

[92] J. Sirignano and K. Spiliopoulos, Mean field analysis of neural networks: a central limit theorem. 2018, arXiv:1808.09372.

[93] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction*. MIT Press, 2018.

[94] V. N. Vapnik and A. Y. Chervonenkis, On the uniform convergence of relative frequencies of events to their probabilities. In *Measures of complexity*, pp. 11–30, Springer, 2015.

[95] C. Villani, *Optimal transport: old and new*. 338, Springer, 2008.

[96] U. Von Luxburg and O. Bousquet, Distance-based classification with Lipschitz functions. *J. Mach. Learn. Res.* **5** (2004), 669–695.

[97] L. Wang, Q. Cai, Z. Yang, and Z. Wang, Neural policy gradient methods: global optimality and rates of convergence. In *International conference on learning representations*, 2019.

[98] Y. Wang, J. Wang, A. Hermann, C. Liu, H. Gao, E. Tosatti, H.-T. Wang, D. Xing, and J. Sun, Electronically driven 1D cooperative diffusion in a simple cubic crystal. *Phys. Rev. X* **11** (2021), no. 1, 011006.

[99] S. Wojtowytsch, On the global convergence of gradient descent training for two-layer ReLU networks in the mean field regime. 2020, arXiv:2005.13530.

[100] S. Wojtowytsch and W. E, Can shallow neural networks beat the curse of dimensionality? a mean field training perspective. *IEEE Trans. Artif. Intell.* **1** (2020), no. 2, 121–129.

[101] S. Wojtowytsch and W. E, Some observations on partial differential equations in Barron and multi-layer spaces. 2020, arXiv:2012.01484.

[102] P. Xie and W. E, Coarse-grained spectral projection: a deep learning assisted approach to quantum unitary dynamics. *Phys. Rev. B* **103** (2021), no. 2, 024304.

[103] Z.-Q. J. Xu, Y. Zhang, T. Luo, Y. Xiao, and Z. Ma, Frequency principle: Fourier analysis sheds light on deep neural networks. 2019, arXiv:1901.06523.

[104] H. Yang and W. E, Generalization and memorization: the bias potential model. 2020, arXiv:2011.14269.

[105] L. Yang and M. Wang, Sample-optimal parametric Q-learning using linearly additive features. In *International conference on machine learning*, pp. 6995–7004, PMLR, 2019.

[106] Z. Yang, C. Jin, Z. Wang, M. Wang, and M. Jordan, Provably efficient reinforcement learning with kernel and neural function approximations. *Adv. Neural Inf. Process. Syst.* **33** (2020).

[107] D. Yarotsky, Error bounds for approximations with deep ReLU networks. *Neural Netw.* **94** (2017), 103–114.

[108] L. Zdeborová and F. Krzakala, Statistical physics of inference: thresholds and algorithms. *Adv. Phys.* **65** (2016), no. 5, 453–552.

[109] J. Zeng, L. Cao, M. Xu, T. Zhu, and J. Z. Zhang, Complex reaction processes in combustion unraveled by neural network-based molecular dynamics simulation. *Nat. Commun.* **11** (2020), no. 1, 1–9.

[110] L. Zhang, J. Han, H. Wang, W. Saidi, R. Car, and W. E, End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems. In *Advances in neural information processing systems 31*, pp. 4441–4451, Curran Associates, Inc., 2018.

[111] L. Zhang, D.-Y. Lin, H. Wang, R. Car, and W. E, Active learning of uniformly accurate interatomic potentials for materials simulation. *Phys. Rev. Mater.* **3** (2019), no. 2, 023804.

[112] L. Zhang, H. Wang, R. Car, and W. E, Phase diagram of a deep potential water model. *Phys. Rev. Lett.* **126** (2021), no. 23, 236001.

[113] L. Zhang, H. Wang, and W. E, Reinforced dynamics for enhanced sampling in large atomic and molecular systems. *J. Chem. Phys.* **148** (2018), no. 12, 124113.

WEINAN E

Center for Machine Learning Research and School of Mathematical Sciences, Peking University, Beijing, China, weinan@math.pku.edu.cn
Beijing Institute for Big Data Research, Beijing, China, weinan@bibdr.org
Department of Mathematics and Program in Applied and Computational Mathematics, Princeton University, Princeton, USA, weinan@math.princeton.edu