Commun. Comput. Phys.
doi: 10.4208/cicp.OA-2020-0185

Vol. **28**, No. 5, pp. 1639-1670
November 2020

Machine Learning and Computational Mathematics

Weinan E[1,2,3,i]

[1] *Department of Mathematics, Princeton University, Princeton, NJ 08544, USA.*
[2] *Program in Applied and Computational Mathematics, Princeton University, Princeton, NJ 08544, USA.*
[3] *Beijing Institute of Big Data Research, Beijing, P.R. China.*

Received 21 September 2020; Accepted (in revised version) 19 October 2020

In memory of Professor Feng Kang (1920-1993)

Abstract. Neural network-based machine learning is capable of approximating functions in very high dimension with unprecedented efficiency and accuracy. This has opened up many exciting new possibilities, not just in traditional areas of artificial intelligence, but also in scientific computing and computational science. At the same time, machine learning has also acquired the reputation of being a set of "black box" type of tricks, without fundamental principles. This has been a real obstacle for making further progress in machine learning.

In this article, we try to address the following two very important questions: (1) How machine learning has already impacted and will further impact computational mathematics, scientific computing and computational science? (2) How computational mathematics, particularly numerical analysis, can impact machine learning? We describe some of the most important progress that has been made on these issues. Our hope is to put things into a perspective that will help to integrate machine learning with computational mathematics.

AMS subject classifications: 65Z99, 62G08, 62H99

Key words: Neural network-based machine learning, machine learning-based algorithm.

1 Introduction

Neural network-based machine learning (ML) has shown very impressive success on a variety of tasks in traditional artificial intelligence. This includes classifying images, generating new images such as (fake) human faces and playing sophisticated games such as

[i]Corresponding author. *Email address:* `weinan@math.princeton.edu` (W. E)

Go. A common feature of all these tasks is that they involve objects in very high dimension. Indeed when formulated in mathematical terms, the image classification problem is a problem of approximating a high dimensional function, defined on the space of images, to the discrete set of values corresponding to the category of each image. The dimensionality of the input space is typically 3 times the number of pixels in the image, where 3 is the dimensionality of the color space. The image generation problem is a problem of generating samples from an unknown high dimensional distribution, given a set of samples from that distribution. The Go game problem is about solving a Bellman-like equation in dynamic programming, since the optimal strategy satisfies such an equation. For sophisticated games such as Go, this Bellman-like equation is formulated on a huge space.

All these are made possible by the ability to accurately approximate high dimensional functions, using modern machine learning techniques. This opens up new possibilities for attacking problems that suffer from the "curse of dimensionality" (CoD): As dimensionality grows, computational cost grows exponentially fast. This CoD problem has been an essential obstacle for the scientific community for a very long time.

Take, for example, the problem of solving partial differential equations (PDEs) numerically. With traditional numerical methods such as finite difference, finite element and spectral methods, we can now routinely solve PDEs in three spatial dimensions plus the temporal dimension. Most of the PDEs currently studied in computational mathematics belong to this category. Well known examples include the Poisson equation, the Maxwell equation, the Euler equation, the Navier-Stokes equations, and the PDEs for linear elasticity. Sparse grids can increase our ability to handling PDEs to, say 8 to 10 dimensions. This allows us to try solving problems such as the Boltzmann equation for simple molecules. But we are totally lost when faced with PDEs, say in 100 dimension. This makes it essentially impossible to solve Fokker-Planck or Boltzmann equations for complex molecules, many-body Schrödinger, or the Hamilton-Jacobi-Bellman equations for realistic control problems.

This is exactly where machine learning can help. Indeed, starting with the work in [10,20,21], machine learning-based numerical algorithm for solving high dimensional PDEs and control problems has been one of the most exciting new developments in recent years in scientific computing, and this has opened up a host of new possibilities for computational mathematics. We refer to [17] for a review of this exciting development.

Solving PDEs is just the tip of the iceberg. There are many other problems for which CoD is the main obstacle, including:

- Classical many-body problem, e.g. protein folding.
- Turbulence. Even though turbulence can be modeled by the three dimensional Navier-Stokes equation, it has so many active degrees of freedom that an effective model for turbulence should involve many variables.
- Solid mechanics. In solid mechanics, we do not even have the analog of the Navier-

Stokes equation. Why is this the case? Well, the real reason is that the behavior of solids is essentially a multi-scale problem that involves scales from atomistic all the way to macroscopic.

- Multi-scale modeling. In fact most multi-scale problems for which there is no separation of scales belong to this category. An immediate example is the dynamics of polymer fluids or polymer melts.

Can machine learning help for these problems? More generally, *can we extend the success of machine learning beyond traditional AI?* We will try to convince the reader that this is indeed the case for many problems.

Besides being extremely powerful, neural network-based machine learning has also got the reputation of being a set of tricks instead of a set of systematic scientific principles. Its performance depends sensitively on the value of the hyper-parameters, such as the network widths and depths, the initialization, the learning rates, etc. Indeed just a few years ago, parameter tuning was considered to be very much of an art. Even now, this is still the case for some tasks. Therefore a natural question is: Can we understand these subtleties and propose better machine learning models whose performance is more robust?

In this article, we review what has been learned on these two issues. We discuss the impact that machine learning has already made or will make on computational mathematics, and how the ideas from computational mathematics, particularly numerical analysis, can be used to help understanding and better formulating machine learning models. On the former, we will mainly discuss the new problems that can now be addressed using ML-based algorithms. Even though machine learning also suggests new ways to solve some traditional problems in computational mathematics, we will not say much on this front.

2 Machine learning-based algorithms for problems in computational science

In this and the next section, we will discuss how neural network models can be used to develop new algorithms. For readers who are not familiar with neural networks, just think of them as being some replacement of polynomials. We will discuss neural networks afterwards.

2.1 Nonlinear multi-grid method and protein folding

In traditional multi-grid method [5], say for solving the linear systems of equation that arise from some finite difference or finite element discretization of a linear elliptic equation, our objective is to minimize a quadratic function like

$$I_h(\boldsymbol{u}_h)=\frac{1}{2}\boldsymbol{u}_h^T L_h \boldsymbol{u}_h - \boldsymbol{f}_h^T \boldsymbol{u}_h.$$

Here h is the grid size of the discretization. The basic idea of the multi-grid method is to iterate between solving this problem and a reduced problem on a coarser grid with grid size H. In order to do this, we need the following:

- A projection operator: $P:\boldsymbol{u}_h\to\boldsymbol{u}_H$, that maps functions defined on the fine grid to functions defined on the coarse grid.
- The effective operator at scale H: $L_H=P^TL_hP$. This defines the objective function on the coarse grid:
$$I_H(\boldsymbol{u}_H)=\frac{1}{2}\boldsymbol{u}_H^TL_H\boldsymbol{u}_H-\boldsymbol{f}_H^T\boldsymbol{u}_H.$$
- A prolongation operator $Q:\boldsymbol{u}_H\to\boldsymbol{u}_h$, that maps functions defined on the coarse grid to functions defined on the fine grid. Usually one can take Q to be P^T.

The key idea here is coarse graining, and iterating between the fine scale and the coarse-grained problem. The main components in coarse graining is a set of coarse-grained variables and the effective coarse-grained problem. Formulated this way, these are obviously general ideas that can be relevant for a wide variety of problems. In practice, however, the difficulty lies in how to obtain the effective coarse-grained problem, a step that is trivial for linear problems, and this is where machine learning can help.

We are going to use the protein folding problem as an example to illustrate the general idea for nonlinear problems.

Let $\{\boldsymbol{x}_j\}$ be the positions of the atoms in a protein and the surrounding solvent, and $U=U(\{\boldsymbol{x}_j\})$ be the potential energy of the combined protein-solvent system. The potential energy consists of the energies due to chemical bonding, Van der Waals interaction, electro-static interaction, etc. The protein folding problem is to find the "ground state" of the energy U:

$$\text{"Minimize" } U.$$

Here we have added quotation marks since really what we want to do is to sample the distribution

$$\rho_\beta=\frac{1}{Z}e^{-\beta U},\quad \beta=(k_BT)^{-1}.$$

Here Z is a normalization factor.

To define the coarse-grained problem, we assume that we are given a set of collective variables: $\boldsymbol{s}=(s_1,\cdots,s_n)$, $s_j=s_j(\boldsymbol{x}_1,\cdots,\boldsymbol{x}_N)$, $(n<N)$. One possibility is to use the dihedral angles as the coarse-grained variables. In principle, one may also use machine learning methods to learn the "best" set of coarse-grained variables but this direction will not be pursued here.

Having defined the coarse-grained variables, the effective coarse-grained problem is simply the free energy associated with this set of coarse-grained variables:

$$A(\boldsymbol{s})=-\frac{1}{\beta}\ln p(\boldsymbol{s}),\quad p_\beta(\boldsymbol{s})=\frac{1}{Z}\int e^{-\beta U(\boldsymbol{x})}\delta(\boldsymbol{s}(\boldsymbol{x})-\boldsymbol{s})\,d\boldsymbol{x},$$

where Z is a normalization factor. Unlike the case for linear problems, for which the effective coarse-grained model is readily available, in the current situation, we have to find the function A first.

The idea is to approximate A by neural networks. The issue here is how to obtain the training data.

Contrary to most standard machine learning problems where the training data is collected beforehand, in applications to computational science and scientific computing, the training data is collected "on-the-fly" as learning proceeds. This is referred to as the "concurrent learning" protocol [11]. In this regard, the standard machine learning problems for which the training data is collected beforehand are examples of "sequential learning". The key issue for concurrent learning is an efficient algorithm for generating the data in the best way. The training dataset should on one hand be representative enough and on the other hand be as small as possible.

A general procedure for generating such datasets is suggested in [11]. It is called the EELT (exploration- examination-labeling-training) algorithm and it consists of the following steps:

- Exploration: exploring the $\boldsymbol{s}$ space. This can be done by sampling $\frac{1}{Z}e^{-\beta A(\boldsymbol{s})}$ with the current approximation of A.

- Examination: for each state explored, decide whether that state should be labeled. One way to do this is to use an *a posteriori error estimator*. One possible such a posteriori error estimator is the variance of the predictions of an ensemble of machine learning models, see [41].

- Labeling: compute the mean force (say using restrained molecular dynamics)

$$\boldsymbol{F}(\boldsymbol{s}) = -\nabla_{\boldsymbol{s}} A(\boldsymbol{s}),$$

  from which the free energy A can be computed using standard thermodynamic integration.

- Training: train the appropriate neural network model. To come up with a good neural network model, one has to take into account the symmetries in the problem. For example, if we coarse grain a full atom representation of a collection of water molecules by eliminating the positions of the hydrogen atoms, then the free energy function for the resulting system should have permutation symmetry and this should be taken into account when designing the neural network model (see the next subsection).

This can also be viewed as a nonlinear multi-grid algorithm in the sense that it iterates between sampling p_β on the space of the coarse-grained variables and the (constrained) Gibbs distribution ρ_β for the full atom description.

This is a general procedure that should work for a large class of nonlinear "multi-grid" problems.

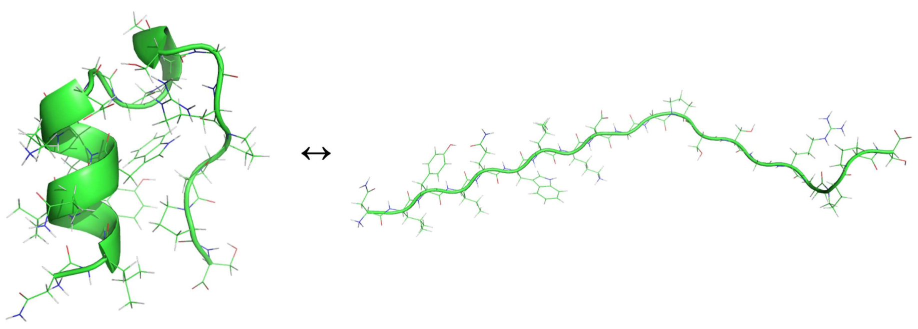


Figure 1: The folded and extended states of Trp-cage, reproduced with permission from [37].

Shown in Fig. 1 is the extended and folded structure of Trp-cage. This is a small protein with 20 amino acids. We have chosen the 38 dihedral angles as the collective variables. The full result is presented in [37].

2.2 Molecular dynamics with *ab initio* accuracy

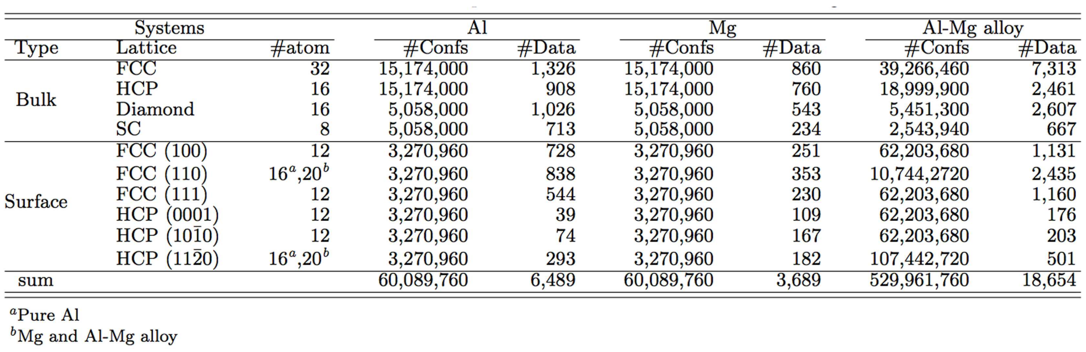
Molecular dynamics is a way of studying the behavior of molecular and material systems by tracking the trajectories of all the nuclei in the system. The dynamics of the nuclei is assumed to obey Newton's law, with some potential energy function (typically called potential energy surface or PES) V that models the effective interaction between the nuclei:

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

How can we get the function V? Traditionally, there have been two rather different approaches. The first is to compute the inter-atomic forces $(-\nabla V)$ on the fly using quantum mechanics models, the most popular one being the density functional theory (DFT). This is known as the Car-Parrinello molecular dynamics or *ab initio* molecular dynamics [6,8]. This approach is quite accurate but is also very expensive, limiting the size of the system that one can handle to about 1000 atoms, even with high performance supercomputers. The other approach is to come up with empirical potentials. Basically one guesses a functional form of V with a small set of fitting parameters which are then determined by a small set of data. This approach is very efficient but unreliable. This dilemma between accuracy and efficiency has been an essential road block for molecular dynamics for a long time.

With machine learning, the new paradigm is to use DFT to generate the data, and then use machine learning to generate an approximation to V. This approach has the potential to produce an approximation to V that is as accurate as the DFT model and as efficient as the empirical potentials.

To achieve this goal, we have to address two issues. The first is the generation of data. The second is coming up with the appropriate neural network model. These two issues



Systems			Al		Mg		Al-Mg alloy	
Type	Lattice	#atom	#Confs	#Data	#Confs	#Data	#Confs	#Data
Bulk	FCC	32	15,174,000	1,326	15,174,000	860	39,266,460	7,313
	HCP	16	15,174,000	908	15,174,000	760	18,999,900	2,461
	Diamond	16	5,058,000	1,026	5,058,000	543	5,451,300	2,607
	SC	8	5,058,000	713	5,058,000	234	2,543,940	667
Surface	FCC (100)	12	3,270,960	728	3,270,960	251	62,203,680	1,131
	FCC (110)	16^a,20^b	3,270,960	838	3,270,960	353	10,744,2720	2,435
	FCC (111)	12	3,270,960	544	3,270,960	230	62,203,680	1,160
	HCP (0001)	12	3,270,960	39	3,270,960	109	62,203,680	176
	HCP $(10\bar{1}0)$	12	3,270,960	74	3,270,960	167	62,203,680	203
	HCP $(11\bar{2}0)$	16^a,20^b	3,270,960	293	3,270,960	182	107,442,720	501
sum			60,089,760	6,489	60,089,760	3,689	529,961,760	18,654

[a]Pure Al
[b]Mg and Al-Mg alloy

Figure 2: The results of EELT algorithm: Number of configurations explored vs. the number of data points labeled. Only a very small percentage of the configurations are labeled. Reproduced with permission from Linfeng Zhang. See also [40].

are the common features for all the problems that we discuss here.

The issue of adaptive data generation is very much the same as before. The EELT procedure can still be used. The details of how each step is implemented is a little different. We refer to [40] for details.

Fig. 2 shows the effect of using the EELT algorithm. As one can see, a very small percentage of the configurations explored are actually labeled. For the Al-Mg example, only $\sim$0.005% configurations explored by are selected for labeling.

For the second issue, the design of appropriate neural networks, the most important considerations are:

1. Extensiveness, the neural network should be extensive in the sense that if we want to extend the system, we just have to extend the neural network accordingly. One way of achieving this is suggested by Behler and Parrinello [4].

2. Preserving the symmetry. Besides the usual translational and rotational symmetry, one also has the permutational symmetry: If we relabel a system of copper atoms, its potential energy should not change. It makes a big difference in terms of the accuracy of the neural network model whether one takes these symmetries into account (see [23] and Fig. 3).

One very nice and general way of addressing the symmetry problem is to design the neural network model as the composition of two networks: An embedding network followed by a fitting network. The task for the embedding network is to represent enough symmetry-preserving functions to be fed into the fitting network [39].

With these issues properly addressed, one can come up with very satisfactory neural network-based representation of V (see Fig. 4). This representation is named Deep Potential [23,39] and the Deep Potential-based molecular dynamics is named DeePMD [38]. As has been demonstrated recently in [25,30], DeePMD, combined with state of the art high performance supercomputers, can help to increase the size of the system that one can model with *ab initio* accuracy by 5 orders of magnitude.

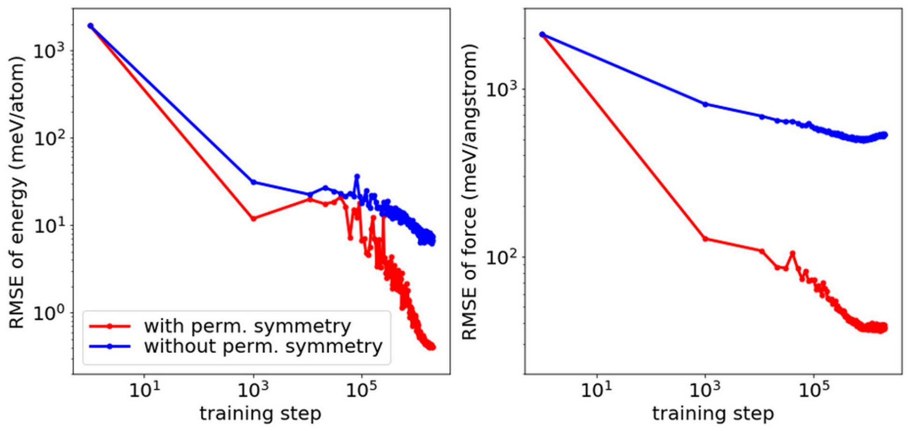


Figure 3: The effect of symmetry preservation on testing accuracy. Shown in red are the results of poor man's way of imposing symmetry (see main text for explanation). One can see that testing accuracy is drastically improved. Reproduced with permission from Linfeng Zhang.

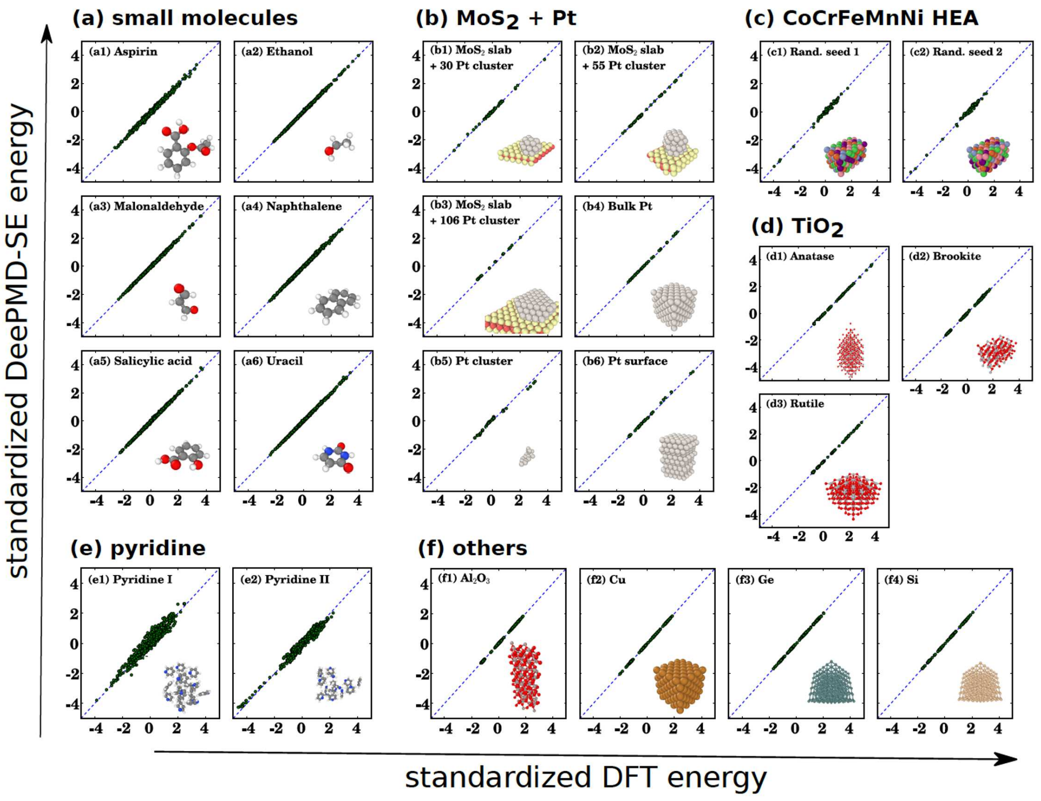


Figure 4: The test accuracy of the Deep Potential for a wide variety of systems. Reproduced with permission from Linfeng Zhang. See also [39].

3 Machine learning-based algorithms for high dimensional problems in scientific computing

3.1 Stochastic control

The first application of machine learning for solving high dimensional problems in scientific computing was presented in [20]. Stochastic control was chosen as the first example due to its close analogy with machine learning. Consider the discrete stochastic dynamical system:

$$s_{t+1}=s_t+b_t(s_t,a_t)+\xi_{t+1}. \tag{3.1}$$

Here s_t and a_t are respectively the state and control at time t, ξ_t is the noise at time t. Our objective is to solve:

$$\min_{\{a_t\}_{t=0}^{T-1}} \mathbb{E}_{\{\xi_t\}}\Big\{\sum_{t=0}^{T-1} c_t(s_t,a_t)+c_T(s_T)\Big\} \tag{3.2}$$

within the set of feedback controls:

$$a_t=A_t(s_t). \tag{3.3}$$

We approximate the functions A_t by neural network models:

$$A_t(s)\approx \tilde{A}_t(s|\theta_t), \quad t=0,\cdots,T-1. \tag{3.4}$$

The optimization problem (3.2) then becomes:

$$\min_{\{\theta_t\}_{t=0}^{T-1}} \mathbb{E}_{\{\xi_t\}}\Big\{\sum_{t=0}^{T-1} c_t(s_t,\tilde{A}_t(s_t|\theta_t))+c_T(s_T)\Big\}. \tag{3.5}$$

Unlike the situation in standard supervised learning, here we have T set of neural networks to be trained simultaneously. The network architecture is shown in Fig. 5.

Compared with the standard setting for machine learning, one can see a clear analogy in which (3.1) plays the role for the residual networks and the noise $\{\xi_t\}$ plays the role of data. Indeed, stochastic gradient descent (SGD) can be readily used to solve the optimization problem (3.5).

An example of the application of this algorithm is shown in Fig. 6 for the problem of energy storage with multiple devices. Here n is the number of devices. For more details, we refer to [20].

3.2 Nonlinear parabolic PDEs

Consider parabolic PDEs of the form:

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

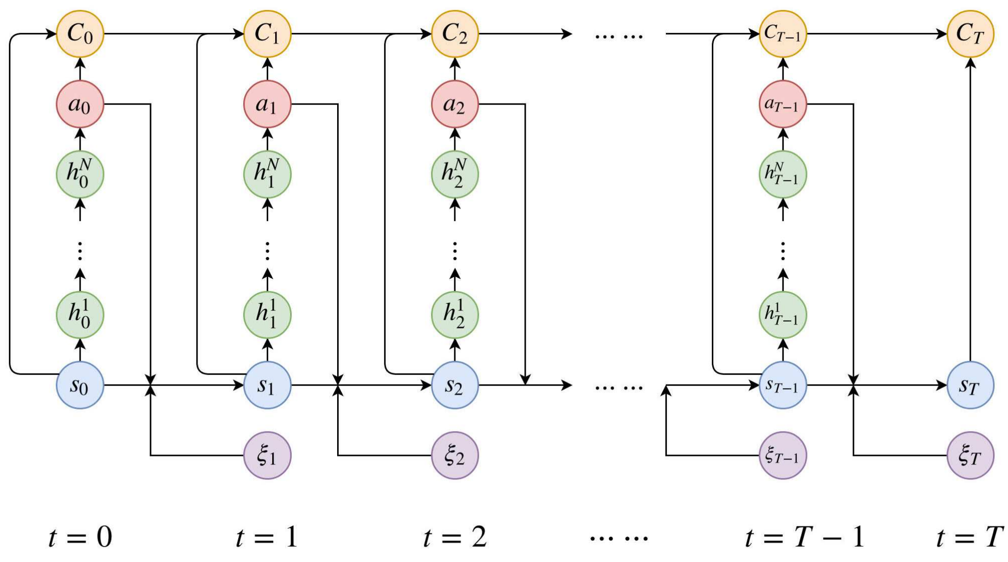


Figure 5: Network architecture for solving stochastic control in discrete time. The whole network has $(N+1)T$ layers in total that involve free parameters to be optimized simultaneously. Each column (except ξ_t) corresponds to a sub-network at t. Reproduced with permission from Jiequn Han. See also [20].

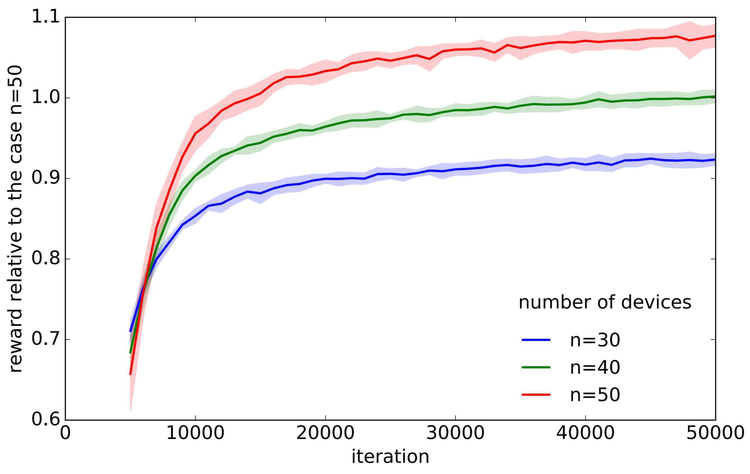


Figure 6: Relative reward for the energy storage problem. The space of control function is $\mathbb{R}^{n+2} \to \mathbb{R}^{3n}$ for $n=30,40,50$, with multiple equality and inequality constrains. Reproduced with permission from Jiequn Han. See also [20].

We study a terminal-value problem instead of initial-value problem since one of the main applications we have in mind is in finance. To develop machine learning-based algorithms, we would like to first reformulate this as a stochastic optimization problem. This can be done using backward stochastic differential equations (BSDE) [33]

$$\inf_{Y_0,\{Z_t\}_{0\le t\le T}} \mathbb{E}|g(X_T)-Y_T|^2, \tag{3.6}$$

$$s.t. \quad X_t=\xi+\int_0^t \mu(s,X_s)\,ds+\int_0^t \Sigma(s,X_s)\,dW_s, \tag{3.7}$$

$$Y_t=Y_0-\int_0^t h(s,X_s,Y_s,Z_s)\,ds+\int_0^t (Z_s)^T dW_s. \tag{3.8}$$

It can be shown that the unique minimizer of this problem is the solution to the PDE with:

$$Y_t=u(t,X_t) \qquad \text{and} \qquad Z_t=\sigma^T(t,X_t)\nabla u(t,X_t). \tag{3.9}$$

With this formulation, one can develop a machine learning-based algorithm along the following lines, adopting the ideas for the stochastic control problems discussed earlier [10,21]:

- After time discretization, approximate the unknown functions

  $$X_0\mapsto u(0,X_0) \qquad \text{and} \qquad X_{t_j}\mapsto \sigma^{\mathrm{T}}(t_j,X_{t_j})\nabla u(t_j,X_{t_j})$$

  by feedforward neural networks ψ and ϕ.

- Using the BSDE, one constructs an approximation $\hat{u}$ that takes the paths $\{X_{t_n}\}_{0\le n\le N}$ and $\{W_{t_n}\}_{0\le n\le N}$ as the input data and gives the final output, denoted by $\hat{u}(\{X_{t_n}\}_{0\le n\le N},\{W_{t_n}\}_{0\le n\le N})$, as an approximation to $u(t_N,X_{t_N})$.

- The error in the matching between $\hat{u}$ and the given terminal condition defines the expected loss function

  $$l(\theta)=\mathbb{E}\Big[\big|g(X_{t_N})-\hat{u}\big(\{X_{t_n}\}_{0\le n\le N},\{W_{t_n}\}_{0\le n\le N}\big)\big|^2\Big].$$

This algorithm is called the Deep BSDE method.

As applications, let us first study a stochastic control problem, but we now solve this problem using the Hamilton-Jacobi-Bellman (HJB) equation. Consider the well-known LQG (linear quadratic Gaussian) problem at dimension $d=100$:

$$dX_t=2\sqrt{\lambda}m_t\,dt+\sqrt{2}\,dW_t, \tag{3.10}$$

with the cost functional: $J(\{m_t\}_{0\le t\le T})=\mathbb{E}\big[\int_0^T \|m_t\|_2^2 dt+g(X_T)\big]$. The corresponding HJB equation is given by

$$\frac{\partial u}{\partial t}+\Delta u-\lambda\|\nabla u\|_2^2=0. \tag{3.11}$$

Using the Hopf-Cole transform, we can express the solution in the form:

$$u(t,x)=-\frac{1}{\lambda}\ln\Big(\mathbb{E}\Big[\exp\Big(-\lambda g(x+\sqrt{2}W_{T-t})\Big)\Big]\Big). \tag{3.12}$$

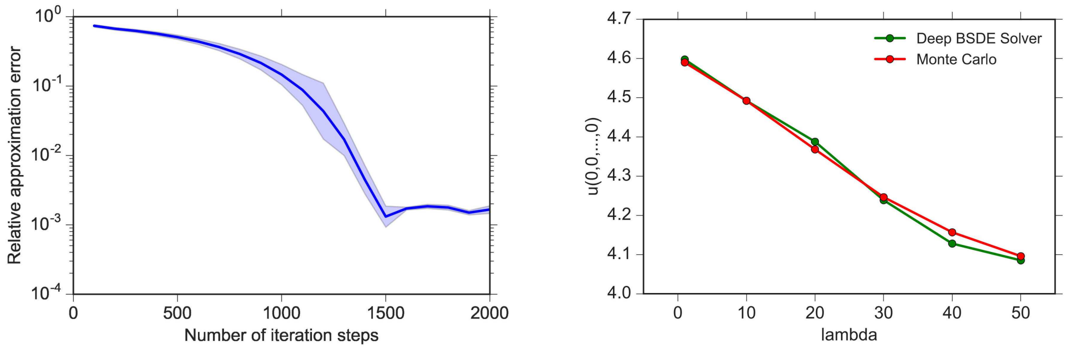


Figure 7: Left: Relative error of the deep BSDE method for $u(t=0,\ x=(0,\cdots,0))$ when $\lambda=1$, which achieves 0.17% in a runtime of 330 seconds. Right: Optimal cost $u(t=0,\ x=(0,\cdots,0))$ against different λ. Reproduced with permission from Jiequn Han. See also [21].

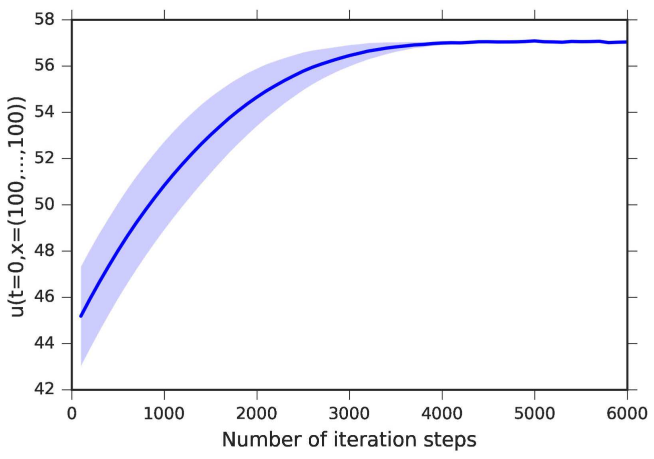


Figure 8: The solution of the Black-Scholes equation with default risk at $d=100$. The Deep BSDE method achieves a relative error of size 0.46% in a runtime of 617 seconds. Reproduced with permission from Jiequn Han. See also [21].

This can be used to calibrate the accuracy of the Deep BSDE method.

As a second example, we study the Black-Scholes equation with default risk:

$$\frac{\partial u}{\partial t}+\Delta u-(1-\delta)Q(u(t,x))u(t,x)-Ru(t,x)=0,$$

where Q is some nonlinear function. This form of modeling the default risk was proposed and/or used in the literature for low dimensional situation ($d=5$, see [21] for references). The Deep BSDE method was used for this problem with $d=100$ [21].

The Deep BSDE method has been applied to pricing basket options, interest rate-dependent options, Libor market model, Bermudan Swaption, barrier option (see [17] for references).

3.3 Moment closure for kinetic equations modeling gas dynamics

The dynamics of gas can be modeled very accurately by the well-known Boltzmann Equation:

$$\partial_t f + \boldsymbol{v}\cdot\nabla_{\boldsymbol{x}} f = \frac{1}{\varepsilon}Q(f), \quad \boldsymbol{v}\in\mathbb{R}^3, \quad \boldsymbol{x}\in\Omega\subset\mathbb{R}^3, \tag{3.13}$$

where f is the phase space density function, ε is the Knudsen number:

$$\varepsilon = \frac{\text{mean free path}}{\text{macroscopic length}},$$

Q is the collision operator that models the collision process between gas particles. When $\varepsilon \ll 1$, this can be approximated by the Euler equation:

$$\partial_t \boldsymbol{U} + \nabla_{\boldsymbol{x}}\cdot \boldsymbol{F}(\boldsymbol{U}) = 0, \tag{3.14}$$

where

$$\boldsymbol{U} = (\rho, \rho\boldsymbol{u}, E)^T, \quad \rho = \int f \mathrm{d}\boldsymbol{v}, \quad \boldsymbol{u} = \frac{1}{\rho}\int f\boldsymbol{v}\mathrm{d}\boldsymbol{v},$$

and

$$\boldsymbol{F}(\boldsymbol{U}) = (\rho\boldsymbol{u}, \rho\boldsymbol{u}\otimes\boldsymbol{u} + pI, (E+p)\boldsymbol{u})^T.$$

Euler's equation can be obtained by projecting the Boltzmann equation on to the low order moments involved, and making use of the ansatz that the distribution function f is close to be local Maxwellian.

What happens when ε is not small? In this case, a natural idea is to seek some generalization of Euler's equation using more moments. This program was initiated by Harold Grad who constructed the well-known 13-moment system using the moments of $\{1, \boldsymbol{v}, (\boldsymbol{v}-\boldsymbol{u})\otimes(\boldsymbol{v}-\boldsymbol{u}), |\boldsymbol{v}-\boldsymbol{u}|^2(\boldsymbol{v}-\boldsymbol{u})\}$. This line of work has encountered several difficulties. First, there is no guarantee that the equations obtained are well-posed. Secondly there is always the "closure problem": When projecting the Boltzmann equation on a set of moments, there are always terms which involve moments outside the set of moments considered. In order to obtain a closed system, one needs to model these terms in some way. For Euler's equation, this is done using the local Maxwellian approximation. This is accurate when ε is small, but is no longer so when ε is not small. It is highly unclear what should be used as the replacement.

In [22], Han et al developed a machine learning-based moment method. The overall objective is to construct an uniformly accurate (generalized) moment model. The methodology consists of two steps:

1: Learn a set of optimal generalized moments through an auto-encoder. Here by optimality we mean that the set of generalized moments retains a maximum amount of information about the original distribution function and can be used to recover the

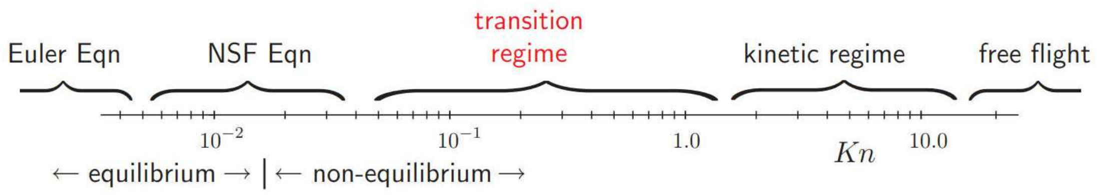


Figure 9: The different regimes of gas dynamics. Reproduced with permission from Jiequn Han. See also [22].

distribution function with a minimum loss of accuracy. This can be done as follows: Find an encoder Ψ and a decoder Φ that recovers the original f from $\boldsymbol{U},\boldsymbol{W}$

$$\begin{aligned}&\boldsymbol{W}=\Psi(f)=\int \boldsymbol{w} f\,\mathrm{d}\boldsymbol{v}, \quad \Phi(\boldsymbol{U},\boldsymbol{W})(\boldsymbol{v})=\boldsymbol{h}(\boldsymbol{v};\boldsymbol{U},\boldsymbol{W}),\\ &\text{Minimize}_{\boldsymbol{w},\boldsymbol{h}} \mathop{\mathbb{E}}_{f\sim\mathcal{D}} \|f-\Phi(\Psi(f))\|^2.\end{aligned}$$

$\boldsymbol{U}$ and $\boldsymbol{W}$ form the set of generalized hydrodynamic variables that we will use to model the gas flow.

2: Learn the fluxes and source terms in the PDE for the projected PDE. The effective PDE for $\boldsymbol{U}$ and $\boldsymbol{W}$ can be obtained by formally projecting the Boltzmann equation on this set of (generalized) moments. This gives us a set of PDEs of the form:

$$\begin{cases}\partial_t \boldsymbol{U}+\nabla_{\boldsymbol{x}}\cdot \boldsymbol{F}(\boldsymbol{U},\boldsymbol{W};\varepsilon)=0,\\ \partial_t \boldsymbol{W}+\nabla_{\boldsymbol{x}}\cdot \boldsymbol{G}(\boldsymbol{U},\boldsymbol{W};\varepsilon)=\boldsymbol{R}(\boldsymbol{U},\boldsymbol{W};\varepsilon),\end{cases} \tag{3.15}$$

where $\boldsymbol{F}(\boldsymbol{U},\boldsymbol{W};\varepsilon)=\int \boldsymbol{v}(1,\boldsymbol{v},\frac{1}{2}|\boldsymbol{v}|^2)^T f d\boldsymbol{v}$, $\boldsymbol{G}(\boldsymbol{U},\boldsymbol{W};\varepsilon)=\int \boldsymbol{v}\boldsymbol{w} f d\boldsymbol{v}$, $\boldsymbol{R}(\boldsymbol{U},\boldsymbol{W};\varepsilon)=\varepsilon^{-1}\int \boldsymbol{w} Q(f) d\boldsymbol{v}$. Our task now is to learn $\boldsymbol{F},\boldsymbol{G},\boldsymbol{R}$ from the original kinetic equation.

Again the important issues are (1) get an optimal dataset, and (2) enforce the physical constraints. Here two notable physical constraints are (1) conservation laws and (2) symmetries. Conservation laws are automatically respected in this approach. Regarding symmetries, besides the usual static symmetry, there is now a new dynamic symmetry: the Galilean invariance. These issues are all discussed in [22]. We also refer to [22] for numerical results for the models obtained this way.

4 Mathematical theory of machine learning

While neural network-based machine learning has demonstrated a wide range of very impressive successes, it has also acquired a reputation of being a "black magic" rather than a solid scientific technique. This is due to the fact that (1) we lack a basic understanding of the fundamental reasons behind its success; (2) the performance of these models and algorithms is quite sensitive to the choice of the hyper-parameters such as the architecture of the network and the learning rate; and (3) some techniques, such as batch normalization, does appear to be a black magic.

To change this situation, we need to (1) improve our understanding of the reasons behind the success and the fragility of neural network-based models and algorithms and (2) find ways to formulate more robust models and design more robust algorithms. In this section we address the first issue. The next section will be devoted to the second issue.

Here is a list of the most basic questions that we need to address:

- Why does it work in such high dimension?
- Why simple gradient descent works for training neural network models?
- Is over-parametrization good or bad?
- Why does neural network modeling require such extensive parameter tuning?

At this point, we do not yet have clear answers to all these questions. But some coherent picture is starting to emerge. We will focus on the problem of supervised learning, namely approximating a target function using a finite dataset. For simplicity, we will limit ourselves to the case when the physical domain of interest is $X=[0,1]^d$.

4.1 An introduction to neural network-based supervised learning

The basic problem in supervised learning is as follows: Given a natural number $n\in\mathbb{N}$ and a sequence $\{(\boldsymbol{x}_j,y_j)=(\boldsymbol{x}_j,f^*(\boldsymbol{x}_j)),\ j\in\{1,2,\cdots,n\}\}$, of pairs of input-output data, we want to recover the target function f^* as accurately as possible. We will assume that the input data $\{\boldsymbol{x}_j, j\in\{1,2,\cdots,n\}\}$, is sampled from the probability distribution μ on $\mathbb{R}^d$.

Step 1. Choose a hypothesis space. This is a set of trial functions $\mathcal{H}_m$ where $m\in\mathbb{N}$ is the dimensionality of $\mathcal{H}_m$. One might choose piecewise polynomials or wavelets. In modern machine learning the most popular choice is neural network functions. Two-layer neural network functions (one input layer, one output layer which usually does not count, and one hidden layer) take the form

$$f_m(\boldsymbol{x},\theta)=\frac{1}{m}\sum_{j=1}^m a_j\sigma(\langle \boldsymbol{w}_j,\boldsymbol{x}\rangle), \tag{4.1}$$

where $\sigma\colon\mathbb{R}\to\mathbb{R}$ is a fixed scalar nonlinear function and where $\theta=\{(a_j,w_j)_{j\in\{1,2,\cdots,m\}}\}$ are the parameters to be optimized (or trained). A popular example for the nonlinear function $\sigma\colon\mathbb{R}\to\mathbb{R}$ is the ReLU (rectified linear unit) activation function: $\sigma(z)=\max\{z,0\}$, for all $z\in\mathbb{R}$. We will restrict our attention to this activation function. Roughly speaking, deep neural network (DNN) functions are obtained if one composes two-layer neural network functions several times. One important class of DNN models are residual neural networks (ResNet). They closely resemble discretized ordinary differential equations and take the form

$$z_{l+1}=z_l+\sum_{j=1}^{M}a_{j,l}\sigma(\langle z_l,w_{j,l}\rangle), \qquad z_0=V\tilde{x}, \qquad f_L(x,\theta)=\langle \alpha,z_L\rangle, \tag{4.2}$$

for $l\in\{0,1,\cdots,L-1\}$ where $L,M\in\mathbb{N}$. Here the parameters are $\theta=(\alpha,V,(a_{j,l}),(w_{j,l}))$. ResNets are the model of choice for truly deep neural network models.

Step 2. Choose a loss function. The primary consideration for the choice of the loss function is to fit the data. Therefore the most obvious choice is the L^2 loss:

$$\hat{\mathcal{R}}_n(f)=\frac{1}{n}\sum_{j=1}^{n}|f(x_j)-y_j|^2=\frac{1}{n}\sum_{j=1}^{n}|f(x_j)-f^*(x_j)|^2. \tag{4.3}$$

This is also called the "empirical risk". Sometimes we also add regularization terms.

Step 3. Choose an optimization algorithm. The most popular optimization algorithms in machine learning are different versions of the gradient descent (GD) algorithm, or its stochastic analog, the stochastic gradient descent (SGD) algorithm. Assume that the objective function we aim to minimize is of the form

$$F(\theta)=\mathbb{E}_{\xi\sim\nu}\big[l(\theta,\xi)\big]. \tag{4.4}$$

The simplest form of SGD iteration takes the form

$$\theta_{k+1}=\theta_k-\eta\nabla l(\theta_k,\xi_k), \tag{4.5}$$

for $k\in\mathbb{N}_0$ where $\{\xi_k,\, k\in\mathbb{N}_0=\{0,1,2,\cdots\}\}$ is a sequence of i.i.d. random variables sampled from the distribution ν and η is the learning rate which might also change during the course of the iteration. In contrast, GD takes the form

$$\theta_{k+1}=\theta_k-\eta\nabla\mathbb{E}_{\xi\sim\nu}\big[l(\theta_k,\xi)\big]. \tag{4.6}$$

Obviously this form of SGD can be adapted to loss functions of the form (4.3) which can be regarded as an expectation with ν being the empirical distribution on the training dataset. This DNN-SGD paradigm is at the heart of modern machine learning.

4.2 Approximation theory

The simplest way of approximating functions is to use polynomials. For polynomial approximation, there are two kinds of theorems. The first is the Weierstrass' Theorem which asserts that continuous functions can be uniformly approximated by polynomials on compact domains. The second is Taylor's Theorem which tells us that the rate of convergence depends on the smoothness of the target function.

Using the terminology in neural network theory, Weierstrass' Theorem is the "Universal Approximation Theorem" (UAT). It is a useful fact. But Taylor's Theorem is more useful since it tells us something about the rate of convergence. The form of Taylor's Theorem used in approximation theory are the direct and inverse approximation theorems which assert that a given function can approximated by polynomials with a particular rate if and only if certain norms of that function is finite. This particular norm, which measures the regularity of the function, is the key quantity that characterizes this approximation scheme. For piecewise polynomials, these norms are some Besov space norms [36]. For L^2, a typical result looks like follows:

$$\inf_{f\in\mathcal{H}_m} \|f-f_m\|_{L^2(X)} \le C_0 h^\alpha \|f\|_{H^\alpha(X)}. \tag{4.7}$$

Here H^α stands for the α-th order Sobolev norm [36], m is the number of degrees of freedom. On a regular grid, the grid size is given by

$$h \sim m^{-1/d}. \tag{4.8}$$

An important thing to notice is that the convergence rate in (4.7) suffers from CoD: If we want to reduce the error by a factor of ϵ, we need to increase m by a factor $m\sim\epsilon^{-d}$ if $\alpha=1$. For $d=100$ which is not very high dimension by the standards of machine learning, this means that we have to increase m by a factor of ϵ^{-100}. This is why polynomials and piecewise polynomials are not useful in high dimensions.

Another way to appreciate this is as follows. The number of monomials of degree p in dimension d is C^d_{p+d}. This grows very fast for large values of d and p.

What should we expect in high dimension? One example that we can learn from is Monte Carlo methods for integration. Consider the problem of approximating

$$I(g) = \mathbb{E}_{\boldsymbol{x}\sim\mu} g(\boldsymbol{x})$$

using

$$I_m(g) = \frac{1}{m}\sum_j g(\boldsymbol{x}_j),$$

where $\{\boldsymbol{x}_j, j\in[m]\}$ is a set of i.i.d samples of the probability distribution μ. A direct computation gives

$$\mathbb{E}(I(g)-I_m(g))^2 = \frac{\operatorname{var}(g)}{m}, \quad \operatorname{var}(g) = \mathbb{E}_{\boldsymbol{x}\sim\mu} g^2(\boldsymbol{x}) - (\mathbb{E}_{\boldsymbol{x}\sim\mu} g(\boldsymbol{x}))^2.$$

This exact relation tells us two things. (1) The convergence rate of Monte Carlo integration is independent of dimension. (2) The error constant is given by the variance of the integrand. Therefore to reduce error, one has to do variance reduction.

Had we used grid-based quadrature rules, the accuracy would have also suffered from CoD.

It is possible to improve the Monte Carlo rate by more sophisticated ways of choosing $\{x_j, j\in[m]\}$, say using number-theoretic-based quadrature rules. But these typically give rise to an $\mathcal{O}(1/d)$ improvement for the convergence rate and it diminishes as $d\to\infty$.

Based on these considerations, we aim to find function approximations that satisfy:

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

The natural questions are then:

- How can we achieve this? That is, what kind of hypothesis space should we choose?
- What should be the "norm" $\|\cdot\|_*$ (associated with the choice of $\mathcal{H}_m$)? Here we put norm in quotation marks since it does not have to be a real norm. All we need is that it controls the approximation error.

Regarding the first question, let us look an illustrative example.

Consider the following Fourier representation of the function f and its approximation f_m (say FFT-based):

$$f(\boldsymbol{x})=\int_{\mathbb{R}^d}a(\boldsymbol{\omega})e^{i(\boldsymbol{\omega},\boldsymbol{x})}d\boldsymbol{\omega},\quad f_m(\boldsymbol{x})=\frac{1}{m}\sum_j a(\boldsymbol{\omega}_j)e^{i(\boldsymbol{\omega}_j,\boldsymbol{x})}.$$

Here $\{\boldsymbol{\omega}_j\}$ is a fixed grid, e.g. uniform grid. For this approximation, we have

$$\|f-f_m\|_{L^2(X)}\le C_0 m^{-\alpha/d}\|f\|_{H^\alpha(X)},$$

which suffers from CoD.

Now consider the alternative representation

$$f(\boldsymbol{x})=\int_{\mathbb{R}^d}a(\boldsymbol{\omega})e^{i(\boldsymbol{\omega},\boldsymbol{x})}\pi(d\boldsymbol{\omega})=\mathbb{E}_{\boldsymbol{\omega}\sim\pi}a(\boldsymbol{\omega})e^{i(\boldsymbol{\omega},\boldsymbol{x})},\tag{4.9}$$

where π is a probability distribution. Now to approximate f, it is natural to use Monte Carlo. Let $\{\boldsymbol{\omega}_j\}$ be an i.i.d. sample of π, $f_m(\boldsymbol{x})=\frac{1}{m}\sum_{j=1}^m a(\boldsymbol{\omega}_j)e^{i(\boldsymbol{\omega}_j,\boldsymbol{x})}$, then we have

$$\mathbb{E}|f(\boldsymbol{x})-f_m(\boldsymbol{x})|^2=\frac{\mathrm{var}(f)}{m}.$$

This approximation does not suffer from CoD. Notice that $f_m(\boldsymbol{x})=\frac{1}{m}\sum_{j=1}^m a_j\sigma(\boldsymbol{\omega}_j^T\boldsymbol{x})$ is nothing but a two-layer neural network with activation function $\sigma(z)=e^{iz}$ (here $a_j=a(\boldsymbol{\omega}_j)$).

We believe that this simple argument is really at the heart of why neural network models do so well in high dimension.

Now let us turn to a concrete example of the kind of approximation theory for neural network models. We will consider two-layer neural networks.

$$\mathcal{H}_m = \Big\{ f_m(\boldsymbol{x}) = \frac{1}{m}\sum_j a_j \sigma(\boldsymbol{w}_j^T \boldsymbol{x}) \Big\}, \quad \theta = \big\{ (a_j, \boldsymbol{w}_j), \, j \in [m] \big\}.$$

Consider function $f: X = [0,1]^d \mapsto \mathbb{R}$ of the following form

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

where $\Omega = \mathbb{R}^1 \times \mathbb{R}^{d+1}$, ρ is a probability distribution on Ω. Define:

$$\|f\|_{\mathcal{B}} = \inf_{\rho \in P_f} \big(\mathbb{E}_\rho [a^2 \|\boldsymbol{w}\|_1^2] \big)^{1/2},$$

where $P_f := \{\rho : f(\boldsymbol{x}) = \mathbb{E}_\rho [a\sigma(\boldsymbol{w}^T\boldsymbol{x})]\}$. This is called the Barron norm [2,12,13]. The space

$$\mathcal{B} = \{ f \in C^0 : \|f\|_{\mathcal{B}} < \infty \}$$

is called the Barron space [2,12,13] (see also [3,14,26]).

In analogy with classical approximation theory, we can also prove some direct and inverse approximation theorem [13].

Theorem 4.1 (Direct Approximation Theorem). *If $\|f\|_{\mathcal{B}} < \infty$, then for any integer $m > 0$, there exists a two-layer neural network function f_m such that*

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

Theorem 4.2 (Inverse Approximation Theorem). *Let*

$$\mathcal{N}_C \stackrel{def}{=} \left\{ \frac{1}{m}\sum_{k=1}^m a_k \sigma(\boldsymbol{w}_k^T \boldsymbol{x}) : \frac{1}{m}\sum_{k=1}^m |a_k|^2 \|\boldsymbol{w}_k\|_1^2 \le C^2, \, m \in \mathbb{N}^+ \right\}.$$

Let f^* be a continuous function. Assume there exists a constant C and a sequence of functions $f_m \in \mathcal{N}_C$ 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 $\boldsymbol{x} \in X$ and $\|f^*\|_{\mathcal{B}} \le C$.

Functions whose Fourier transforms decay sufficiently fast satisfy the conditions of this theorem [13].

4.3 Estimation error

Another issue we have to worry about is the performance of the machine learning model outside the training dataset. This issue also shows up in classical approximation theory. Illustrated in Fig. 10 is the classical Runge phenomenon for polynomial interpolation on a uniform grid. One can see that away from the grid points, the error of the interpolant can be very large. This is a situation that we would like to avoid.

What we do in practice is to minimize the training error:

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

but we are interested in the testing error, which is a sampled version of the population risk:

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

The question is how we can control the difference between these two errors.

One way of doing this is to use the notion of Rademacher complexity. The important fact for us here is that the Rademacher complexity controls the difference between training and testing errors (also called the "generalization gap"). Indeed, let $\mathcal{H}$ be a set of functions, and $S=(\boldsymbol{x}_1,\boldsymbol{x}_2,\cdots,\boldsymbol{x}_n)$ be a dataset. Then, up to logarithmic terms, we have

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

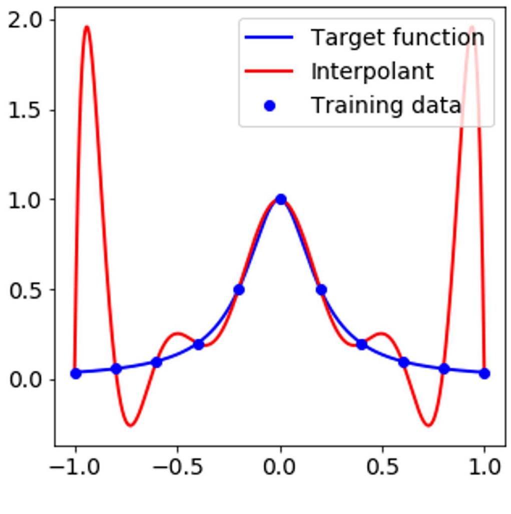


Figure 10: The Runge phenomenon: $f^*(x)=\frac{1}{1+25x^2}$. Reproduced with permission from Chao Ma.

where the **Rademacher complexity** of $\mathcal{H}$ with respect to S is defined as

$$\mathrm{Rad}_S(\mathcal{H}) = \frac{1}{n}\mathbb{E}_\xi\left[\sup_{h\in\mathcal{H}}\sum_{i=1}^{n}\xi_i h(\boldsymbol{x}_i)\right], \tag{4.10}$$

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

The question then becomes to bound the Rademacher complexity of a hypothesis space. For the Barron space, we have [2]:

Theorem 4.3. *Let $\mathcal{F}_Q = \{f\in\mathcal{B}, \|f\|_\mathcal{B}\le Q\}$. Then we have*

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

where $n=|S|$, the size of the dataset S.

4.4 A priori estimates for regularized models

Consider the regularized model

$$\mathcal{L}_n(\theta) = \hat{\mathcal{R}}_n(\theta) + \lambda\sqrt{\frac{\log(2d)}{n}}\|\theta\|_\mathcal{P}, \qquad \hat{\theta}_n = \operatorname{argmin}\,\mathcal{L}_n(\theta), \tag{4.11}$$

where the path norm is defined by:

$$\|\theta\|_\mathcal{P} = \left(\frac{1}{m}\sum_{k=1}^{m}|a_k|^2\|\boldsymbol{w}_k\|_1^2\right)^{1/2}.$$

The following result was proved in [12]:

Theorem 4.4. *Assume $f^*: X\mapsto[0,1]\in\mathcal{B}$. There exist constants C_0, such that for any $\delta>0$, if $\lambda\ge C_0$, then with probability at least $1-\delta$ over the choice of the training dataset, we have*

$$\mathcal{R}(\hat{\theta}_n)\lesssim\frac{\|f^*\|_\mathcal{B}^2}{m}+\lambda\|f^*\|_\mathcal{B}\sqrt{\frac{\log(2d)}{n}}+\sqrt{\frac{\log(1/\delta)+\log(n)}{n}}.$$

Similar approximation theory and a priori error estimates have been proved for other machine learning models. Here is a brief summary of these results.

- Random feature model: The corresponding function space is the reproducing kernel Hilbert space (RKHS).
- Residual networks (ResNets): The corresponding function space is the so-called flow-induced space introduced in [13].

- Multi-layer neural networks: A candidate for the corresponding function space is the multi-layer space introduced in [15].

What is really important is the "norms" that control the approximation error and the generalization gap. These quantities are defined for functions in the corresponding spaces. After the approximation theorems and Rademacher complexity estimates are in place, one can readily prove a theorem of the following type for regularized models: Up to logarithmic terms, the minimizers of the regularized models satisfy:

$$\mathcal{R}(\hat{f}) \lesssim \frac{\Gamma(f^*)}{m} + \frac{\gamma(f^*)}{\sqrt{n}},$$

where m is the number of free parameters, n is the size of the training dataset. Note that for the multilayer spaces, the results proved in [15] are not as sharp.

We only discussed the analysis of the hypothesis space. There are many more other questions. We refer to [19] for more discussion on the current understanding of neural network-based machine learning.

5 Machine learning from a continuous viewpoint

Now we turn to alternative formulations of machine learning. Motivated by the situation for PDEs, we would like to first formulate machine learning in a continuous setting and then discretize to get concrete models and algorithms. The key here is that continuous problems that we come up with should be nice mathematical problems. For PDEs, this is accomplished by requiring them to be "well-posed". For problems in calculus of variations, we require the problem to be "convex" in some sense and lower semi-continuous. The point of these requirements is to make sure that the problem has a unique solution. Intuitively, for machine learning problems, being "nice" means that the variational problem should have a simple landscape. How to formulate this precisely is an important research problem for the future.

As was pointed out in [16], the key ingredients for the continuous formulation are as follows:

- representation of functions (as expectations);
- formulating the variational problem (as expectations);
- optimization, e.g. gradient flows.

5.1 Representation of functions

Two kinds of representations are considered in [16]: Integral transform-based representation and flow-based representation. The simplest integral-transform based representation is a generalization of (4.9):

$$\begin{aligned}f(\boldsymbol{x};\theta)=&\int_{\mathbb{R}^d}a(\boldsymbol{w})\sigma(\boldsymbol{w}^T\boldsymbol{x})\pi(d\boldsymbol{w})\\=&\mathbb{E}_{\boldsymbol{w}\sim\pi}a(\boldsymbol{w})\sigma(\boldsymbol{w}^T\boldsymbol{x})\\=&\mathbb{E}_{(a,\boldsymbol{w})\sim\rho}a\sigma(\boldsymbol{w}^T\boldsymbol{x})\\=&\mathbb{E}_{\boldsymbol{u}\sim\rho}\phi(\boldsymbol{x},\boldsymbol{u}).\end{aligned}$$

Here θ denotes the parameters in the model: θ can be $a(\cdot)$ or the prob distributions π or ρ.

This representation corresponds to two-layer neural networks. A generalization to multi-layer neural networks is presented in [15].

Next we turn to flow-based representation:

$$\frac{d\boldsymbol{z}}{d\tau}=\mathbb{E}_{\boldsymbol{w}\sim\pi_\tau}\boldsymbol{a}(\boldsymbol{w},\tau)\sigma(\boldsymbol{w}^T\boldsymbol{z}) \tag{5.1}$$

$$=\mathbb{E}_{(\boldsymbol{a},\boldsymbol{w})\sim\rho_\tau}\boldsymbol{a}\sigma(\boldsymbol{w}^T\boldsymbol{z}) \tag{5.2}$$

$$=\mathbb{E}_{\boldsymbol{u}\sim\rho_\tau}\boldsymbol{\phi}(\boldsymbol{z},\boldsymbol{u}),\quad \boldsymbol{z}(0,\boldsymbol{x})=\boldsymbol{x}, \tag{5.3}$$

$$f(\boldsymbol{x},\theta)=\mathbf{1}^T\boldsymbol{z}(1,\boldsymbol{x}).$$

In this representation, the parameter θ can be either $\{a_\tau(\cdot)\}$ or $\{\pi_\tau\}$ or $\{\rho_\tau\}$.

5.2 The stochastic optimization problem

Stochastic optimization problems are of the type:

$$\min_\theta \mathbb{E}_{\boldsymbol{w}\sim\nu}g(\theta,\boldsymbol{w}).$$

These kinds of problems can readily be approached by stochastic algorithms, which is a key component in machine learning. For example, instead of the gradient descent algorithm:

$$\theta_{k+1}=\theta_k-\eta\nabla_\theta\mathbb{E}_{\boldsymbol{w}\sim\nu}g(\theta,\boldsymbol{w})=\theta_k-\eta\nabla_\theta\sum_{j=1}^n g(\theta,\boldsymbol{w}_j),$$

one can use the stochastic gradient descent:

$$\theta_{k+1}=\theta_k-\eta\nabla_\theta g(\theta,\boldsymbol{w}_k),$$

where $\{\boldsymbol{w}_k\}$ is a set of random variables sampled from ν.

The following are some examples of the stochastic optimization problems that arise in modern machine learning:

- Supervised learning: In this case, the minimization of the population risk becomes

$$\mathcal{R}(f)=\mathbb{E}_{\boldsymbol{x}\sim\mu}(f(\boldsymbol{x})-f^*(\boldsymbol{x}))^2.$$

- Eigenvalue problems for quantum many-body Hamiltonian:

$$I(\phi)=\frac{(\phi,H\phi)}{(\phi,\phi)}=\mathbb{E}_{x\sim\mu_\phi}\frac{\phi(\boldsymbol{x})H\phi(\boldsymbol{x})}{\phi(\boldsymbol{x})^2},\quad \mu_\phi(d\boldsymbol{x})=\frac{1}{Z}|\phi(\boldsymbol{x})|^2 d\boldsymbol{x}.$$

  Here H is the Hamiltonian of the quantum system.

- Stochastic control problems:

$$L(\{a_t\}_{t=0}^{T-1})=\mathbb{E}\left\{\sum_{t=0}^{T-1}c_t(s_t,a_t))+c_T(s_T)\right\}.$$

Substituting the representations discussed earlier to these expressions for the stochastic optimization problems, we obtain the final variational problem that we need to solve.

One can either discretize these variational problems directly and then solve the discretized problem using some optimization algorithms, or one can write down continuous forms of some optimization algorithms, typically gradient flow dynamics, and then discretize these continuous flows. We are going to discuss the second approach.

5.3 Optimization: Gradient flows

To write continuous form of the gradient flows, we draw some inspiration from statistical physics. Take the supervised learning as an example. We regard the population risk as being the "free energy", and following Halperin and Hohenberg [24], we divide the parameters into two kinds, conserved and non-conserved. For example, a is a non-conserved parameter and π is conserved since its total integral has to be 1.

For non-conserved parameter, as was suggested in [24], one can use the "model A" dynamics:

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

which is simply the usual L^2 gradient flow.

For conserved parameters such as π, one should use the "model B" dynamics which works as follows: First define the "chemical potential"

$$V=\frac{\delta\mathcal{R}}{\delta\pi}.$$

From the chemical potential, one obtains the velocity field $\boldsymbol{v}$ and the current J:

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

The continuity equation then gives us the gradient flow dynamics:

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

This is also the gradient flow under the Wasserstein metric.

5.4 Discretizing the gradient flows

To obtain practical models, one needs to discretize these continuous problems. The first step is to replace the population risk by the empirical risk using the training data.

The more non-trivial issue is how to discretize the gradient flows in the parameter space. The parameter space has the following characteristics: (1) It has a simple geometry – unlike the real space which may have a complicated geometry. (2) It is also usually high dimensional. For these reasons, the most natural numerical methods for the discretization in the parameter space is the particle method which is the dynamic version of Monte Carlo. Smoothed particle method might be helpful to improve the performance. In relatively low dimensions, one might also consider the spectral method, particularly some sparse version of the spectral method, due to the relative simple geometry of the parameter space.

Take for example the discretization of the conservative flow for the integral transform-based representation. With the representation: $f(\boldsymbol{x};\theta)=\mathbb{E}_{(a,\boldsymbol{w})\sim\rho}a\sigma(\boldsymbol{w}^T\boldsymbol{x})$, the gradient flow equation becomes:

$$\partial_t\rho=\nabla(\rho\nabla V),\quad V=\frac{\delta\mathcal{R}}{\delta\rho}. \tag{5.4}$$

The particle method discretization is based on:

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

where $\boldsymbol{u}_j(t)=(a_j(t),\boldsymbol{w}(t))$. One can show that in this case, (5.4) reduces to

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

where

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

This is exactly gradient descent for "scaled" two-layer neural networks. In this case, the continuous formulation also coincides with the "mean-field" formulation for two-layer neural networks [7,32,34,35].

The scaling factor $1/m$ in front of the "scaled" two-layer neural networks is actually quite important and makes a big difference for the test performance of the network. Shown in Fig. 11 is the heat map of the test error for two-layer neural network models with and without this scaling factor. The target function is the simple single neuron function: $f^*(\boldsymbol{x})=\sigma(x_1)$. The important observation is that in the absence of this scaling factor, the test error shows a "phase transition" between a phase where the neural network model performs like an associated random feature model and another phase where it shows much better performance than the random feature model. Such a phase transition is one of the reasons that choosing the right set of hyper-parameters, here the network

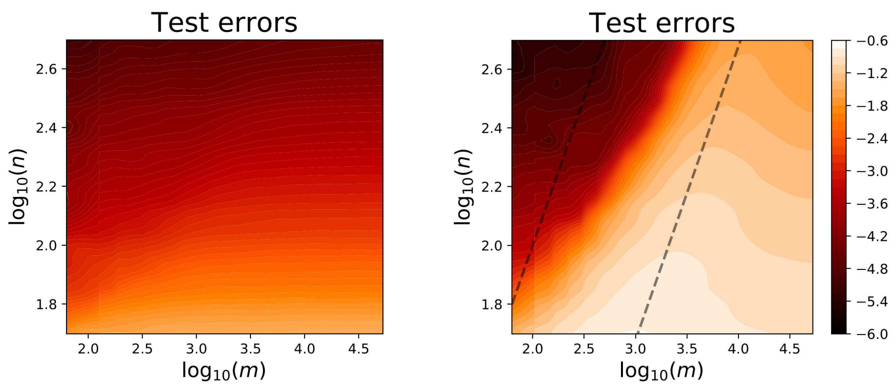


Figure 11: (Left) continuous viewpoint; (Right) conventional NN models. Target function is a single neuron. Reproduced with permission from Lei Wu.

width m, is so important. However, if one uses the scaled form, this phase transition phenomenon is avoided and the performance is more robust [31].

5.5 The optimal control problem for flow-based representation

The flow-based representation naturally leads to a control problem. This viewpoint has been used explicitly or implicitly in machine learning for quite some time (see for example [27]). The back propagation algorithm, for example, is an example of control-theory based algorithm. Another more recent example is the development of maximum principle-based algorithm, first introduced in [28]. Despite these successes, we feel that there is still a lot of room for using the control theory viewpoint to develop new algorithms.

We consider the flow-based representation in a slightly more general form

$$\frac{d\boldsymbol{z}}{d\tau} = \mathbb{E}_{\boldsymbol{u}\sim\rho_\tau}\boldsymbol{\phi}(\boldsymbol{z},\boldsymbol{u}), \quad \boldsymbol{z}(0,\boldsymbol{x}) = \boldsymbol{x},$$

where $\boldsymbol{z}$ is the state, ρ_τ is the control at "time" τ. Our objective is to minimize $\mathcal{R}$ over $\{\rho_\tau\}$

$$\mathcal{R}(\{\rho_\tau\}) = \mathbb{E}_{\boldsymbol{x}\sim\mu}(f(\boldsymbol{x}) - f^*(\boldsymbol{x}))^2 = \int_{\mathbb{R}^d}(f(\boldsymbol{x}) - f^*(\boldsymbol{x}))^2 d\mu,$$

where as before

$$f(\boldsymbol{x}) = \mathbf{1}^T\boldsymbol{z}(1,\boldsymbol{x}). \tag{5.5}$$

One most important result for this control problem is Pontryagin's maximum principle (PMP). To state this result, let us define the Hamiltonian $H: \mathbb{R}^d \times \mathbb{R}^d \times \mathcal{P}_2(\Omega) :\mapsto \mathbb{R}$ as

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

Pontryagin's maximum principle asserts that the solutions of the control problem must satisfy:

$$\rho_\tau = \operatorname{argmax}_\rho \mathbb{E}_{\boldsymbol{x}}[H(\boldsymbol{z}_\tau^{t,x}, \boldsymbol{p}_\tau^{t,x}, \rho)], \quad \forall \tau \in [0,1], \tag{5.6}$$

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

$$\begin{aligned} \frac{d\boldsymbol{z}_\tau^{t,x}}{d\tau} &= \nabla_{\boldsymbol{p}} H = \mathbb{E}_{\boldsymbol{u}\sim\rho_\tau(\cdot;t)}[\boldsymbol{\phi}(\boldsymbol{z}_\tau^{t,x}, \boldsymbol{u})], \\ \frac{d\boldsymbol{p}_\tau^{t,x}}{d\tau} &= -\nabla_{\boldsymbol{z}} H = \mathbb{E}_{\boldsymbol{u}\sim\rho_\tau(\cdot;t)}[\nabla_{\boldsymbol{z}}^T \boldsymbol{\phi}(\boldsymbol{z}_\tau^{t,x}, \boldsymbol{u}) \boldsymbol{p}_\tau^{t,x}], \end{aligned} \tag{5.7}$$

with the boundary conditions:

$$\boldsymbol{z}_0^{t,x} = \boldsymbol{x}, \tag{5.8}$$

$$\boldsymbol{p}_1^{t,x} = -2(f(\boldsymbol{x};\rho(\cdot;t)) - f^*(\boldsymbol{x}))\mathbf{1}. \tag{5.9}$$

Pontryagin's maximum principle is slightly stronger than the KKT condition for the stationary points in that (5.6) is a statement of optimality rather than criticality. In fact (5.6) also holds when the parameters are discrete and this has been used in [29] to develop efficient numerical algorithms for this case.

With the help of PMP, it is also easy to write down the gradient descent flow for the optimization problem. Formally, one can simply write down the gradient descent flow for (5.6) for each τ:

$$\partial_t \rho_\tau(\boldsymbol{u},t) = \nabla \cdot (\rho_\tau(\boldsymbol{u},t) \nabla V(\boldsymbol{u};\rho)), \quad \forall \tau \in [0,1], \tag{5.10}$$

where

$$V(\boldsymbol{u};\rho) = \mathbb{E}_{\boldsymbol{x}}\Big[\frac{\delta H}{\delta \rho}(\boldsymbol{z}_\tau^{t,x}, \boldsymbol{p}_\tau^{t,x}, \rho_\tau(\cdot;t))\Big],$$

and $\{(\boldsymbol{z}_\tau^{t,x}, \boldsymbol{p}_\tau^{t,x})\}$ are defined as before by the forward/backward equations.

To discretize the gradient flow, we can simply use:

- forward Euler for the flow in τ variable, with step size $1/L$ (L is the number of grid points in the τ variable);
- particle method for the gradient descent dynamics, with M samples in each τ-grid point.

This gives us

$$\boldsymbol{z}_{l+1}^{t,x} = \boldsymbol{z}_l^{t,x} + \frac{1}{LM}\sum_{j=1}^{M} \boldsymbol{\phi}(\boldsymbol{z}_l^{t,x}, \boldsymbol{u}_l^j(t)), \quad l = 0,\cdots,L-1, \tag{5.11}$$

$$\boldsymbol{p}_l^{t,x} = \boldsymbol{p}_{l+1}^{t,x} + \frac{1}{LM}\sum_{j=1}^{M} \nabla_{\boldsymbol{z}} \boldsymbol{\phi}(\boldsymbol{z}_{l+1}^{t,x}, \boldsymbol{u}_{l+1}^j(t)) \boldsymbol{p}_{l+1}^{t,x}, \quad l = 0,\cdots,L-1, \tag{5.12}$$

$$\frac{d\boldsymbol{u}_l^j(t)}{dt} = -\mathbb{E}_{\boldsymbol{x}}\big[\nabla_{\boldsymbol{w}}^T \boldsymbol{\phi}(\boldsymbol{z}_l^{t,x}, \boldsymbol{u}_l^j(t)) \boldsymbol{p}_l^{t,x}\big]. \tag{5.13}$$

This recovers the GD algorithm (with back-propagation) for the (scaled) ResNet:

$$\boldsymbol{z}_{l+1}=\boldsymbol{z}_l+\frac{1}{LM}\sum_{j=1}^{M}\boldsymbol{\phi}(\boldsymbol{z}_l,\boldsymbol{u}_l).$$

We call this "scaled" ResNet because of the presence of the factor $1/(LM)$.

In a similar spirit, one can also obtain an algorithm using PMP [28]. Adopting the terminology in control theory, this kind of algorithms are called "method of successive approximation" (MSA). The basic MSA is as follows:

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

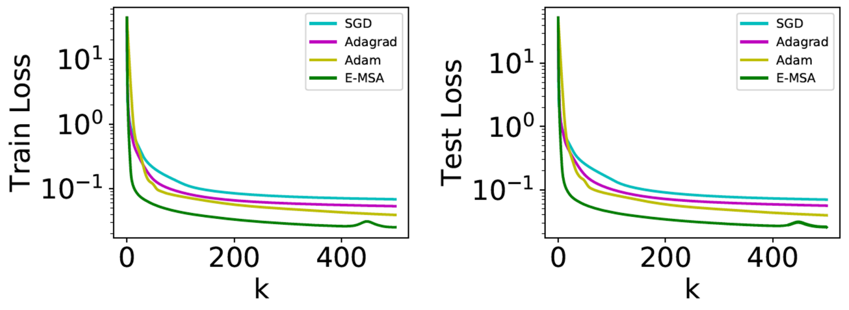
In practice, this basic version does not perform as well as the "extended MSA" which works in the same way as the MSA except that the Hamiltonian is replaced by an extended Hamiltonian [28]:

$$\tilde{H}(\boldsymbol{z},\boldsymbol{p},\theta,\boldsymbol{v},\boldsymbol{q}):=H(\boldsymbol{z},\boldsymbol{p},\theta)-\frac{1}{2}\lambda\|\boldsymbol{v}-f(\boldsymbol{z},\theta)\|^2-\frac{1}{2}\lambda\|\boldsymbol{q}+\nabla_{\boldsymbol{z}}H(\boldsymbol{z},\boldsymbol{p},\theta)\|^2.$$

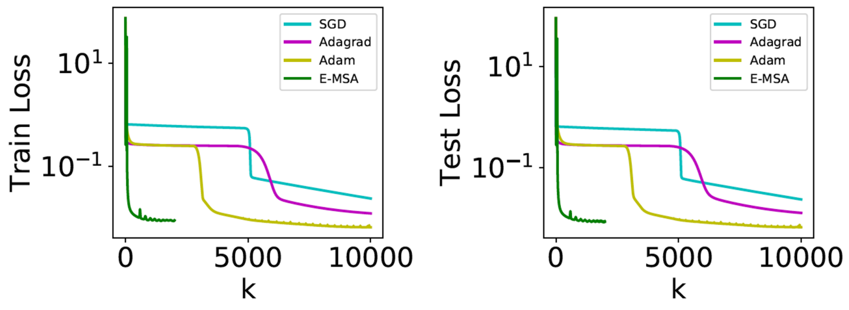
Fig. 12 shows the results of the extended MSA compared with different versions of SGD for two kinds of initialization. One can see that in terms of the number of iterations, extended MSA outperforms all the SGDs. In terms of wall clock time, the advantage of the extended MSA is diminished significantly. This is possibly due to the inefficiencies in the implementation of the optimization algorithm (here the BFGS) used for solving (5.6). We refer to [28] for more details. In any case, it is clear that there is a lot of room for improvement.

6 Concluding remarks

In conclusion, we have discussed a wide range of problems for which machine learning-based algorithms have made and/or will make a significant difference. These problems



(a)



(b)

Figure 12: Comparison of the extended MSA with different versions of stochastic gradient descent algorithms. The top figures show results for small initialization. The bottom figures show results for bigger initialization. Reproduced with permission from Qianxiao Li. See also [28].

are relatively new to computational mathematics. We believe strongly that machine learning-based algorithms will also significantly impact the way we solve more traditional problems in computational mathematics. However, research in this direction is still at a very early stage.

Another important area that machine learning can be of great help is multi-scale modeling. The moment-closure problem discussed above is an example in this direction. There are many more possible applications, see [8]. Machine learning seems to be able to provide the missing link in making advanced multi-scale modeling techniques really practical. For example in the heterogeneous multi-scale method (HMM) [1,9], one important component is to extract the relevant macro-scale information from micro-scale simulation data. This step has always been a major obstacle in HMM. It seems quite clear that machine learning techniques can of great help here.

We also discussed how the viewpoint of numerical analysis can help to improve the mathematical foundation of machine learning as well as propose new and possibly more robust formulations. In particular, we have given a taste of how high dimensional approximation theory should look like. We also demonstrated that commonly used machine learning models and training algorithms can be recovered from some particular discretization of continuous models, in a scaled form. From this discussion, one can see that neural network models are quite natural and rather inevitable.

What have we really learned from machine learning? Well, it seems that the most important new insight from machine learning is the representation of functions as expectations. We reproduce them here for convenience:

- Integral-transform based:

$$f(\boldsymbol{x}) = \mathbb{E}_{(a,\boldsymbol{w})\sim\rho} a\sigma(\boldsymbol{w}^T\boldsymbol{x}),$$
$$f(\boldsymbol{x}) = \mathbb{E}_{\theta_L\sim\pi_L} a^{(L)}_{\theta_L}\sigma(\mathbb{E}_{\theta_{L-1}\sim\pi_{L-1}}\cdots\sigma(\mathbb{E}_{\theta_1\sim\pi_1} a^1_{\theta_2,\theta_1}\sigma(a^0_{\theta_1}\cdot\boldsymbol{x}))\cdots);$$

- Flow-based:

$$\frac{d\boldsymbol{z}}{d\tau} = \mathbb{E}_{(\boldsymbol{a},\boldsymbol{w})\sim\rho_\tau} \boldsymbol{a}\sigma(\boldsymbol{w}^T\boldsymbol{z}), \quad \boldsymbol{z}(0,\boldsymbol{x}) = \boldsymbol{x}, \tag{6.1}$$
$$f(\boldsymbol{x},\theta) = \mathbf{1}^T\boldsymbol{z}(1,\boldsymbol{x}). \tag{6.2}$$

From the viewpoint of computational mathematics, this suggests that the central issue will move from specific discretization schemes to more effective representations of functions.

This review is rather sketchy. Interested reader can consult the three review articles [17–19] for more details.

Acknowledgments

I am very grateful to my collaborators for their contribution to the work described here. In particular, I would like to express my sincere gratitude to Roberto Car, Jiequn Han, Arnulf Jentzen, Qianxiao Li, Chao Ma, Han Wang, Stephan Wojtowytsch, and Lei Wu for the many discussions that we have had on the issues discussed here. This work is supported in part by a gift to the Princeton University from iFlytek as well as the ONR grant N00014-13-1-0338.

References

[1] A. Abdulle, W. E, B. Engquist and E. Vanden-Eijnden, The heterogenous multiscale methods, Acta Numerica, 21:1-87, 2012.
[2] F. Bach, Breaking the curse of dimensionality with convex neural networks, Journal of Machine Learning Research, 18(19):1-53, 2017.
[3] A. R. Barron, Universal approximation bounds for superpositions of a sigmoidal function, IEEE Transactions on Information theory, 39(3):930-945, 1993.
[4] J. Behler and M. Parrinello, Generalized neural-network representation of high-dimensional potential-energy surfaces, Physical review letters, 98(14):146401, 2007.

[5] A. Brandt, Multiscale scientific computation: Review 2001. In: Barth, T.J., Chan, T.F. and Haimes, R. (eds.): Multiscale and Multiresolution Methods: Theory and Applications, Springer Verlag, Heidelberg, 2001, pp. 1-96.

[6] R. Car and M. Parrinello, Unified approach for molecular dynamics and density-functional theory, Physical Review Letters, 55(22):2471, 1985.

[7] 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.

[8] W. E, Principles of Multiscale Modeling, Cambridge University Press, 2011.

[9] W. E and B. Engquist, The heterogeneous multiscale methods, Comm. Math. Sci., 1(1):87-132, 2003.

[10] W. E, J. Han and A. Jentzen, Deep learning-based numerical methods for high-dimensional parabolic partial differential equations and backward stochastic differential equations, Communications in Mathematics and Statistics, 5(4):349-380, 2017.

[11] W. E, J. Han and L. Zhang, Integrating Machine Learning with Physics-Based Modeling, https://arxiv.org/pdf/2006.02619.pdf, 2020.

[12] W. E, C. Ma and L. Wu, A priori estimates of the population risk for two-layer neural networks, Communications in Mathematical Sciences, 17(5):1407-1425, 2019.

[13] W. E, C. Ma and L. Wu, Barron spaces and the flow-induced function spaces for neural network models, arXiv:1906.08039, 2019.

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

[15] W. E and S. Wojtowytsch, On the Banach spaces associated with multi-layer ReLU networks: Function representation, approximation theory and gradient descent dynamics, https://arxiv.org/pdf/2007.15623.pdf, 2020.

[16] W. E, C. Ma, and L. Wu, Machine learning from a continuous viewpoint, arXiv:1912.12777, 2019.

[17] W. E, J. Han and A. Jentzen, Algorithms for solving high dimensional PDEs: From nonlinear Monte Carlo to machine learning, https://arxiv.org/pdf/2008.13333.pdf, 2020.

[18] W. E, J. Han and L. Zhang, Integrating machine learning with physics-based modeling, https://arxiv.org/pdf/2006.02619.pdf, 2020.

[19] W. E, C. Ma, S. Wojtowytsch and L. Wu, Towards a mathematical understanding of machine learning: What is known and what is not, https://arxiv.org/pdf/2009.10713.pdf.

[20] J. Han and W. E, Deep learning approximation for stochastic control problems, Deep Reinforcement Learning Workshop, NIPS (2016), https://arxiv.org/pdf/1611.07422.pdf.

[21] J. Han, A. Jentzen and W. E, Solving high-dimensional partial differential equations using deep learning, Proceedings of the National Academy of Sciences, 115(34):8505-8510, 2018.

[22] J. Han, C. Ma, Z. Ma and W. E, Uniformly accurate machine learning based hydrodynamic models for kinetic equations, Proceedings of the National Academy of Sciences, 116(44): 21983-21991; DOI: 10.1073/pnas.1909854116, 2019.

[23] J. Han, L. Zhang, R. Car and W. E, Deep potential: A general representation of a many-body potential energy surface, Communications in Computational Physics, 23(3):629-639, 2018.

[24] P. C. Hohenberg and B. I. Halperin, Theory of dynamic critical phenomena, Reviews of Modern Physics, 49(3):435, 1977.

[25] W. Jia, H. Wang, M. Chen, D. Lu, J. Liu, 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, arXiv:2005.00223, 2020.

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

[27] Y. LeCun, A theoretical framework for back propagation, in: Touretzky, D., Hinton, G., Sejnouski, T. (eds.), Proceedings of the 1988 connectionist models summer school, Carnegie-Mellon University, Morgan Kaufmann, 1989.

[28] Q. Li, L. Chen, C. Tai and W. E, Maximum principle based algorithms for deep learning, Journal of Machine Learning Research, 18(165):1-29, 2018, https://arxiv.org/pdf/1710.09513.pdf.

[29] Q. Li and S. Hao, An optimal control approach to deep learning and applications to discrete-weight neural networks, Proceedings of the 35th International Conference on Machine Learning, 2018.

[30] D. Lu, H. Wang, M. Chen, J. Liu, L. Lin, R. Car, W. E, W. Jia and L. Zhang, 86 pflops deep potential molecular dynamics simulation of 100 million atoms with ab initio accuracy, arXiv:2004.11658, 2020.

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

[32] S. Mei, A. Montanari and P.-M. Nguyen, A mean field view of the landscape of two-layer neural networks, Proceedings of the National Academy of Sciences, 115(33):E7665-E7671, 2018.

[33] E. Pardoux and S. Peng, Backward stochastic differential equations and quasilinear parabolic partial differential equations, in: Stochastic Partial Differential Equations and Their Applications (Charlotte, NC, 1991), vol. 176 of Lecture Notes in Control and Inform. Sci., Springer, Berlin, 1992, pp. 200-217.

[34] 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.

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

[36] H. Tribel, Theory of Function Spaces, Birkhäuser, 1983.

[37] H. Wang, L. Zhang and W. E, unpublished.

[38] L. Zhang, J. Han, H. Wang, R. Car and W. E, Deep potential molecular dynamics: A scalable model with the accuracy of quantum mechanics, Physical Review Letters, 120:143001, Apr 2018.

[39] L. Zhang, J. Han, H. Wang, W. A. Saidi, R. Car and W. E, End-to-end symmetry preserving inter-atomic potential energy model for finite and extended systems, in: Advances of the Neural Information Processing Systems (NIPS), 2018.

[40] L. Zhang, D.-Y. Lin, H. Wang, R. Car and W. E, Active learning of uniformly accurate inter-atomic potentials for materials simulation, Physical Review Materials, 3(2):023804, 2019.

[41] L. Zhang, H. Wang and W. E, Reinforced dynamics for the enhanced sampling in large atomic and molecular systems. I. Basic methodology, J. Chem. Phys., 148:124113, 2018.