Machine Learning and Computational Mathematics

机器学习与计算数学

Weinan E (鄂维南)

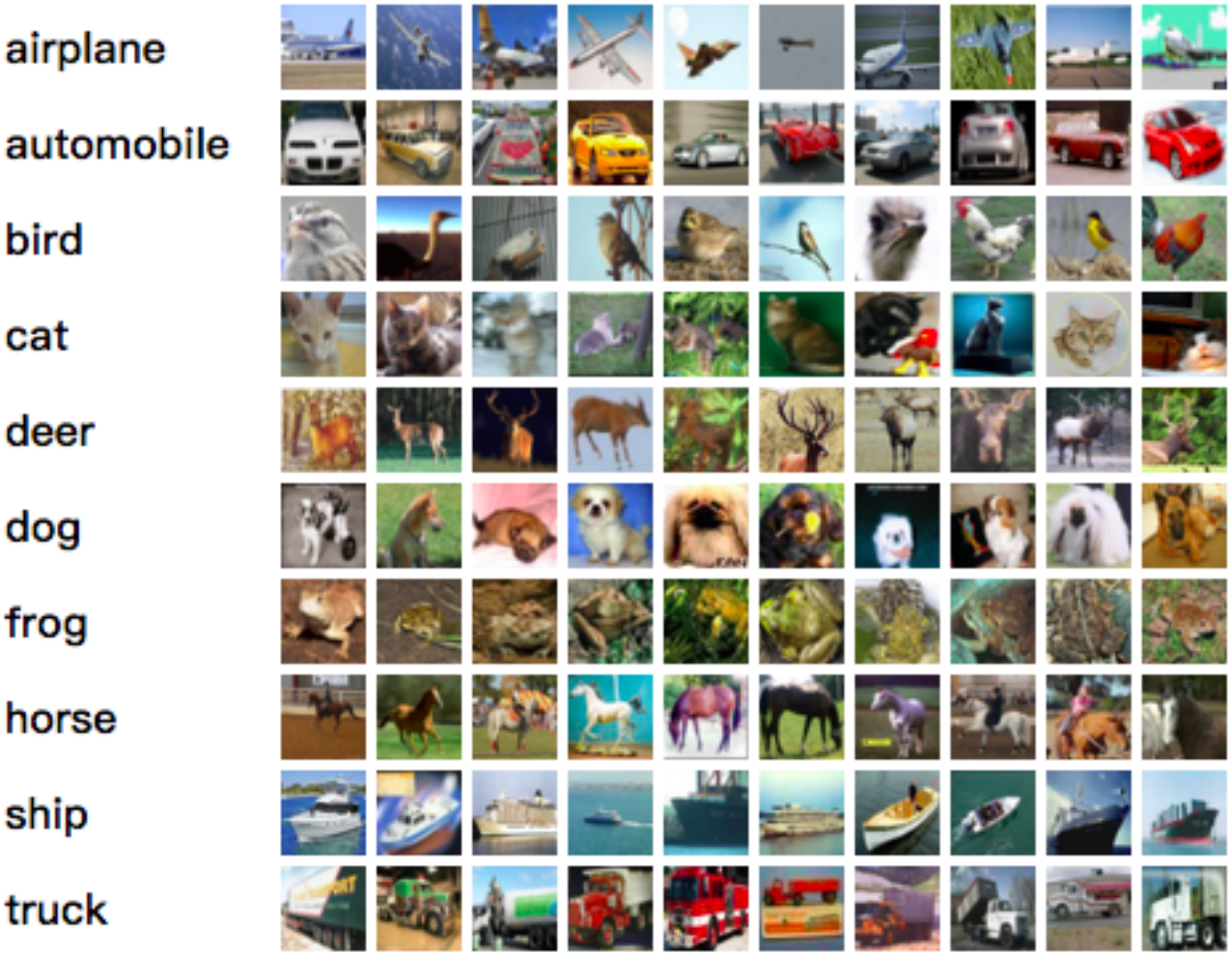
- How machine learning **will** impact scientific computing and computational science?
- How computational mathematics **can** impact machine learning?

- 机器学习会给计算数学带来什么样的改变?
- 计算数学能够怎样帮助推进机器学习的研究?

ML can do wonders: Approximating high dimensional functions

Given $S = \{(x_j, y_j = f^*(x_j)), j \in [n]\}$, learn (i.e. approximate) f^*.

Example: Cifar 10 dataset (f^* is a discrete function defined on the space of images)



- Input: each image $\in [0,1]^d$, $d = 32 \times 32 \times 3 = 3072$.
- Output: $f^* \in \{$airplane, ... , truck$\}$.
- $f^* : [0,1]^{3072} \rightarrow \{$airplane, ... , truck$\}$.
  f^*(each image) = category

Sampling unknown high dimensional distributions

The probability distribution of all real and fake human faces is an unknown distribution in high dimension.

Solving high dimensional Bellman equations

The optimal strategy obeys some Bellman-like equation.

All these are made possible by our ability to accurately approximate high dimensional functions using finite pieces of data.

This opens up new possibilities for attacking problems that suffer from the "curse of dimensionality" (CoD):

As dimensionality grows, computational cost grows exponentially fast.

CoD: Solving PDEs using traditional numerical methods

- usually $d = 1, 2, 3, 4$
  - handles Poisson, Maxwell, Euler, Navier-Stokes, elasticity, etc. well.
- sparse grids: $d \sim 10$
  - we can barely solve Boltzmann for simple molecules.
- $d \geq 100$, impossible
  - can't deal with realistic control problems, Fokker-Planck or Boltzmann for complex molecules, many-body Schrödinger, etc.

This is where machine learning can help.

Han and E (2016, NIPS workshop), E, Han and Jentzen (2017, Comm Math Stat), Han, Jentzen and E (2018, PNAS)

Other examples of problems that face CoD

- quantum many-body problems
- classical many-body problem, e.g. protein folding
- turbulence
- solid mechanics (plasticity, nonlinear elasticity)
- control
- .....
- multi-scale modeling (gas dynamics, combustion, non-Newtonian fluids, etc)

Can machine learning help?

Can the success of ML be extended beyond traditional AI ?

Analogy with the situation for finite element method (FEM)

FEM: structural mechanics $\rightarrow$ general PDEs

ML: traditional AI $\rightarrow$ general function approximation problems

Functions are basic tools in mathematics.

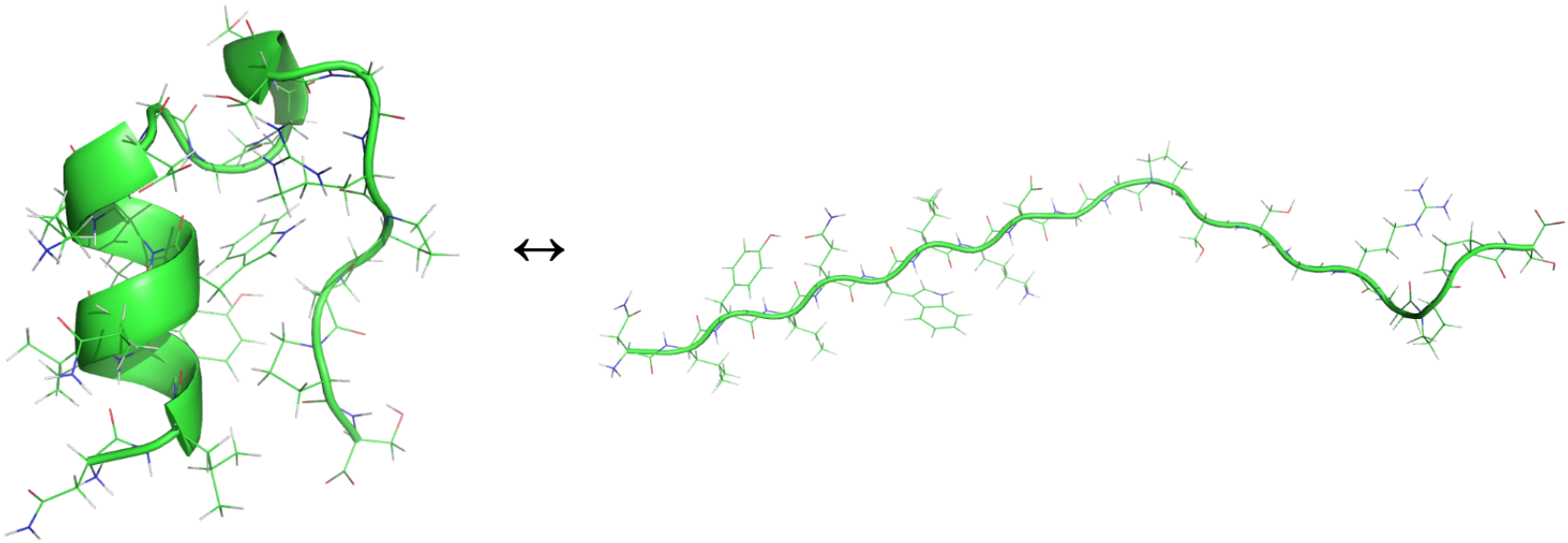
Imagine that we have something like "polynomials" but works in high dimension!

Example 1: Protein folding

$\{\boldsymbol{x}_j\}$= positions of the atoms in a protein
$U(\{\boldsymbol{x}_j\})$ = potential energy (chemical bonding, Van der Waals, electro-static, etc).

$$\text{“Minimize” } U, \quad \text{or sample } \rho = \frac{1}{Z} e^{-\beta U},\ \beta = (k_B T)^{-1}$$

Folding Trp-cage (20 amino acids, 38 collective variables)



Han Wang, Linfeng Zhang, Weinan E (2018)

Multi-grid viewpoint of protein folding

Traditional multi-grid: Minimize $I_h(\boldsymbol{u}_h) = \frac{1}{2}\boldsymbol{u}_h^T L_h \boldsymbol{u}_h - \boldsymbol{f}_h^T \boldsymbol{u}_h$

- projection operator: $P : \boldsymbol{u}_h \to \boldsymbol{u}_H$
- effective operator on scale H: $L_H = P^T L_h P$.
  - effective problem on scale H: Minimize $I_H(\boldsymbol{u}_H) = \frac{1}{2}\boldsymbol{u}_H^T L_H \boldsymbol{u}_H - \boldsymbol{f}_H^T \boldsymbol{u}_H$

Multi-grid approach to protein folding: "Minimize" $U(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots \boldsymbol{x}_N)$

- collective variables: $\boldsymbol{s} = (s_1, \cdots, s_n), s_j = s_j(\boldsymbol{x}_1, \cdots, \boldsymbol{x}_N), (n < N)$
- effective energy = free energy:

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

  - effective problem on coarser scale: Minimize $A(\boldsymbol{s}) = A(s_1, s_2, \cdots, s_n)$

Now we have to find the function A first!

- A will be represented by neural networks.
- data comes from "accelerated" molecular dynamics, using **adaptive** sampling:

  generate a sequence of samples $\{\boldsymbol{s}_j\}$ adaptively, and use them the train a more and more accurate neural network approximation to A.

The EELT (exploration- examination-labeling-training) algorithm (Zhang et al (2018)):

- exploring the $\boldsymbol{s}$ space, say by sampling $\frac{1}{Z}e^{-\beta A(\boldsymbol{s})}$ with the current approximation of A.
- for each state explored, decide whether that state should be labeled, say use an *a posteriori error estimator*.
- labeling: compute the mean force (using restrained MD)
- training: deep potential-like coarse-grained model

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

2. DeePMD: Molecular dynamics with *ab initio* accuracy

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

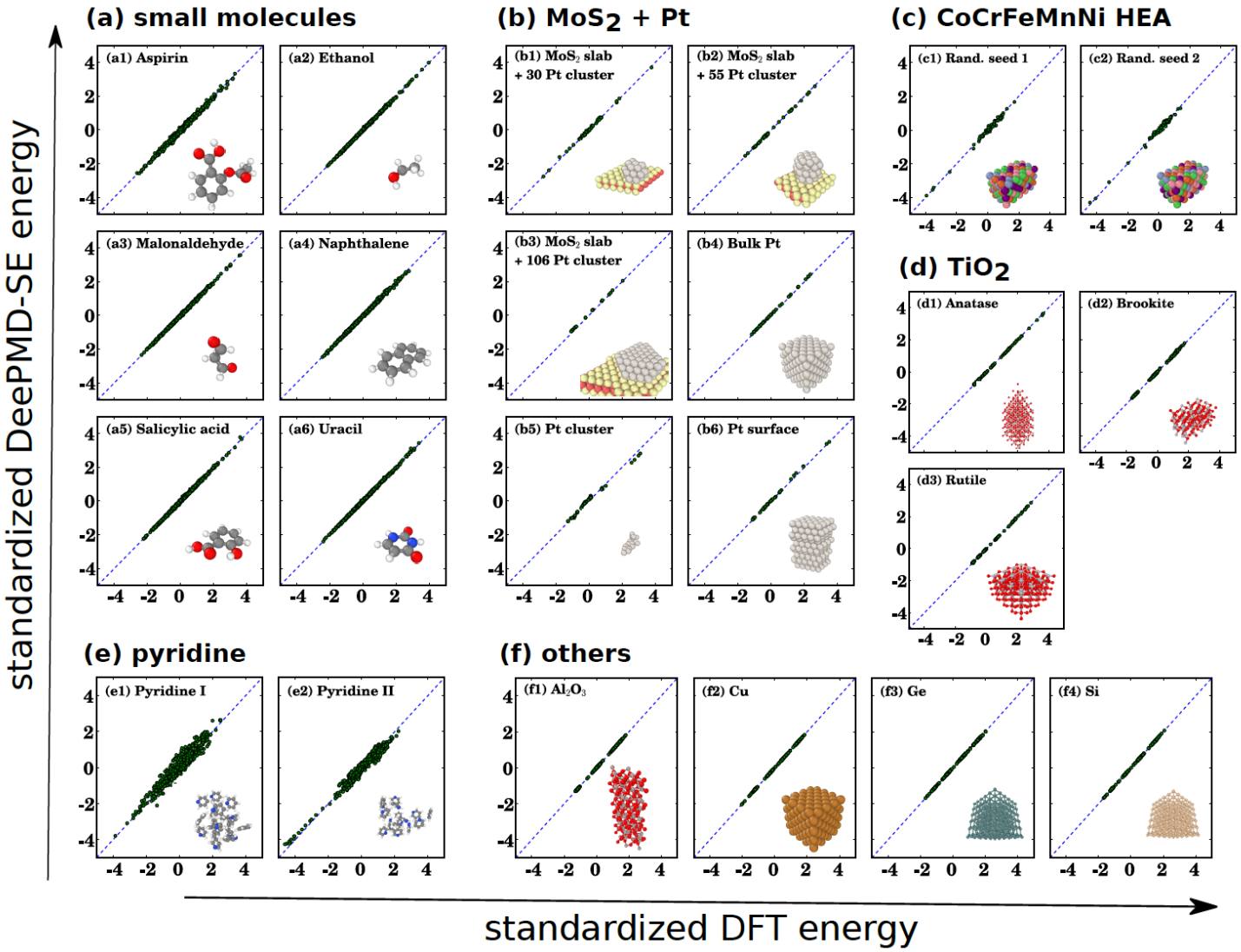
Key question: $V = ?$

Two ways to calculate V:

- Computing the inter-atomic forces on the fly using QM, e.g. the Car-Parrinello MD.
  Accurate but expensive (limited to about 1000 atoms).
- Empirical potentials: basically guess what V should be.
  Efficient but unreliable.

Now: Use QM to supply the data needed to train a neural network model.

Accuracy comparable to QM for a wide range of materials and molecules



Linfeng Zhang, Jiequn Han, et al (2018)

Adaptive generation of data: The EELT algorithm

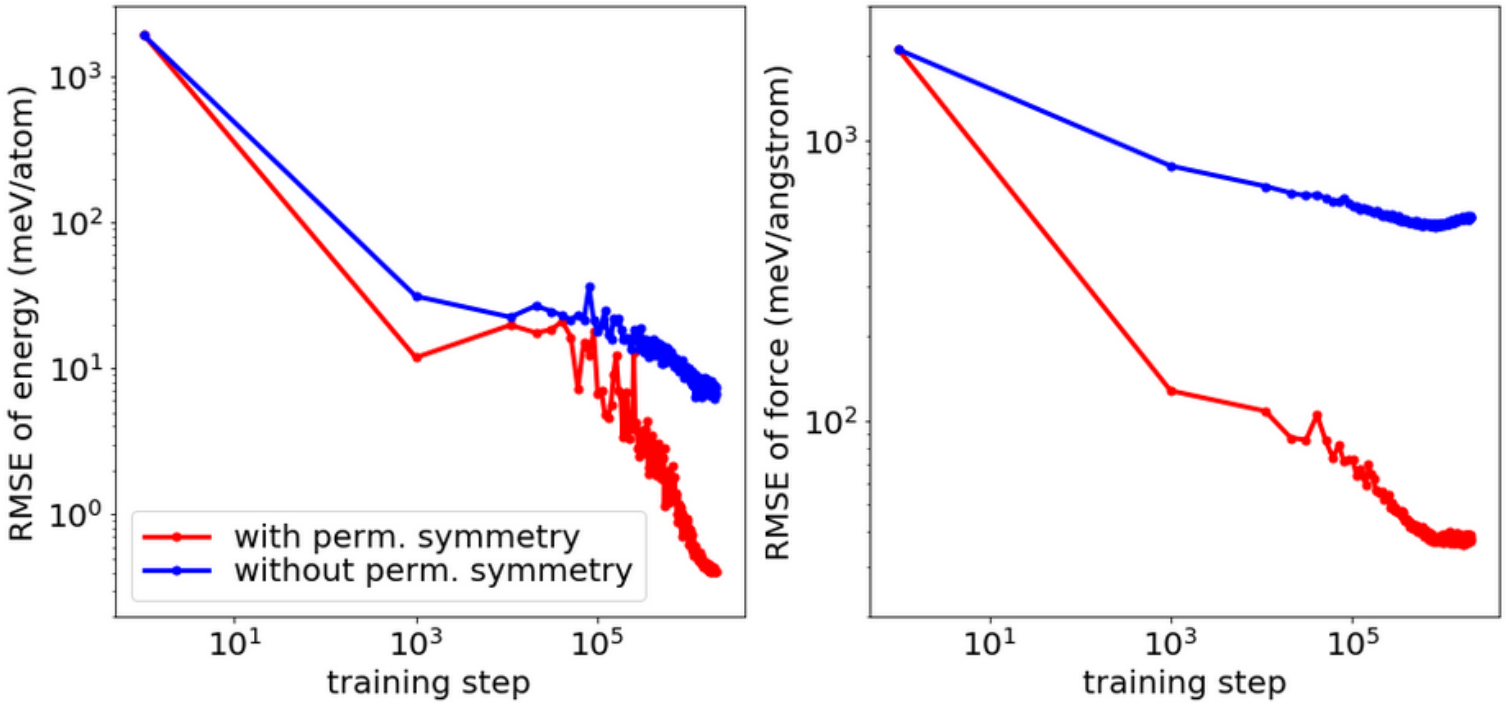
Systems			Al		Mg		Al-Mg alloy	
Type	Lattice	#atom	#Confs	#Data	#Confs	#Data	#Confs	#Data
Bulk	FCC	32	15,174,000	1,326	15,174,000	860	39,266,460	7,313
	HCP	16	15,174,000	908	15,174,000	760	18,999,900	2,461
	Diamond	16	5,058,000	1,026	5,058,000	543	5,451,300	2,607
	SC	8	5,058,000	713	5,058,000	234	2,543,940	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

$\sim$0.005% configurations explored by DeePMD are selected for labeling.

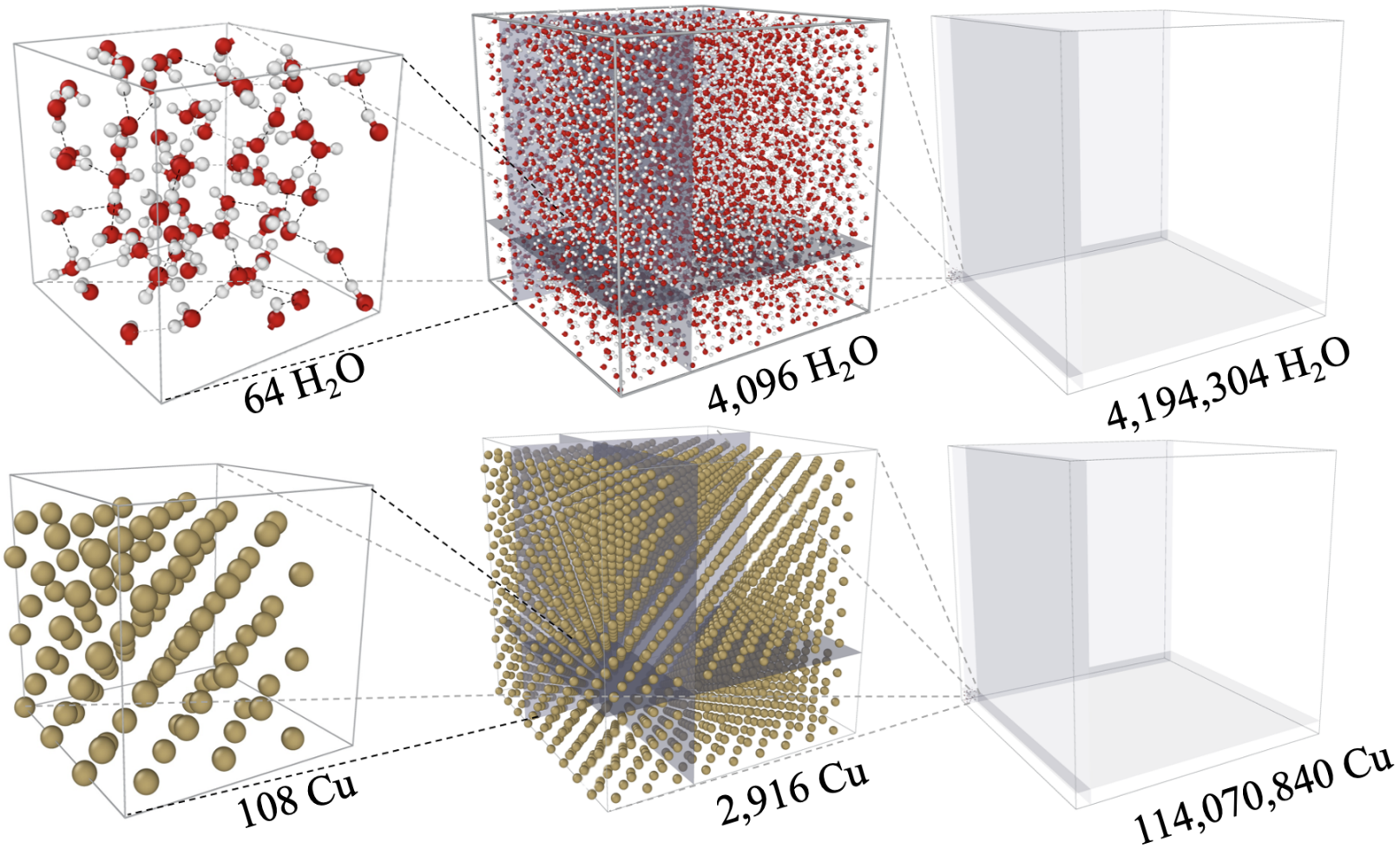
Linfeng Zhang, Han Wang, et al (2019)

The importance of preserving the symmetries



Jiequn Han, Linfeng Zhang, Roberto Car and Weinan E (2017)

DeePMD simulation of 100M atoms with *ab initio* accuracy



(a) AIMD + HPC;

(b) DeePMD+1 GPU @ Home;

(c) DeePMD+27360 GPUs @ Summit.

D. Lu, et al, arXiv: 2004.11658; W. Jia, et al, arXiv: 2005.00223

3. Coarse grained (CG) MD – A basic tool for chemical and biological engineering

Traditional approach: Bead and spring models, no accuracy
DeePCG: ML-based models (combine the ideas mentioned above)

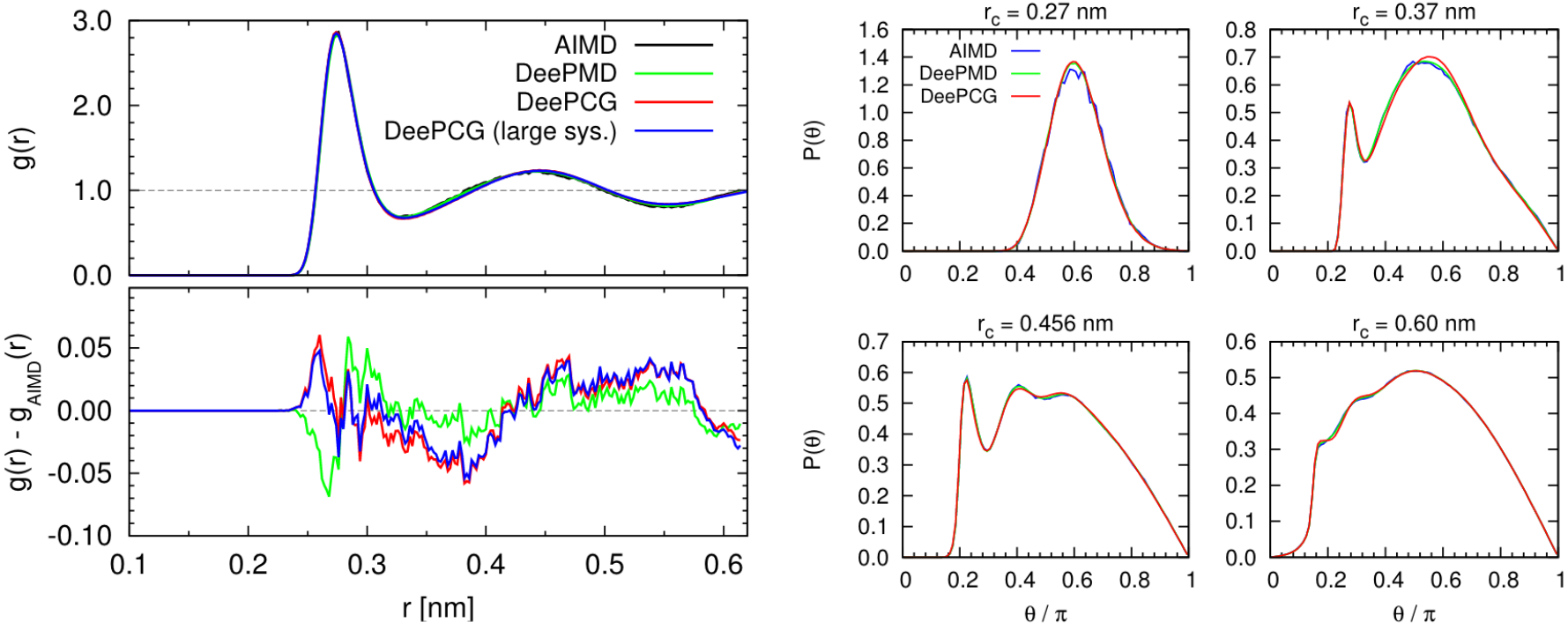


Figure: Radial distribution functions (left) and angular distribution functions (right)

L. Zhang, J. Han, H. Wang, R. Car and W. E, J. Chem. Phys. 2018

4. Stochastic control (Han and E (2016))

Model dynamics (analog of ResNet):

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

s_t = state, a_t = control, ξ_t = noise.

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

Look for a feedback control:

$$a_t = a_t(s_t).$$

- Neural network approximation:

$$a_t(s_t) \approx \tilde{a}_t(s_t|\theta_t),\ t = 0, \cdots, T-1$$

Optimization problem (SGD applies directly)

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

Network Architecture

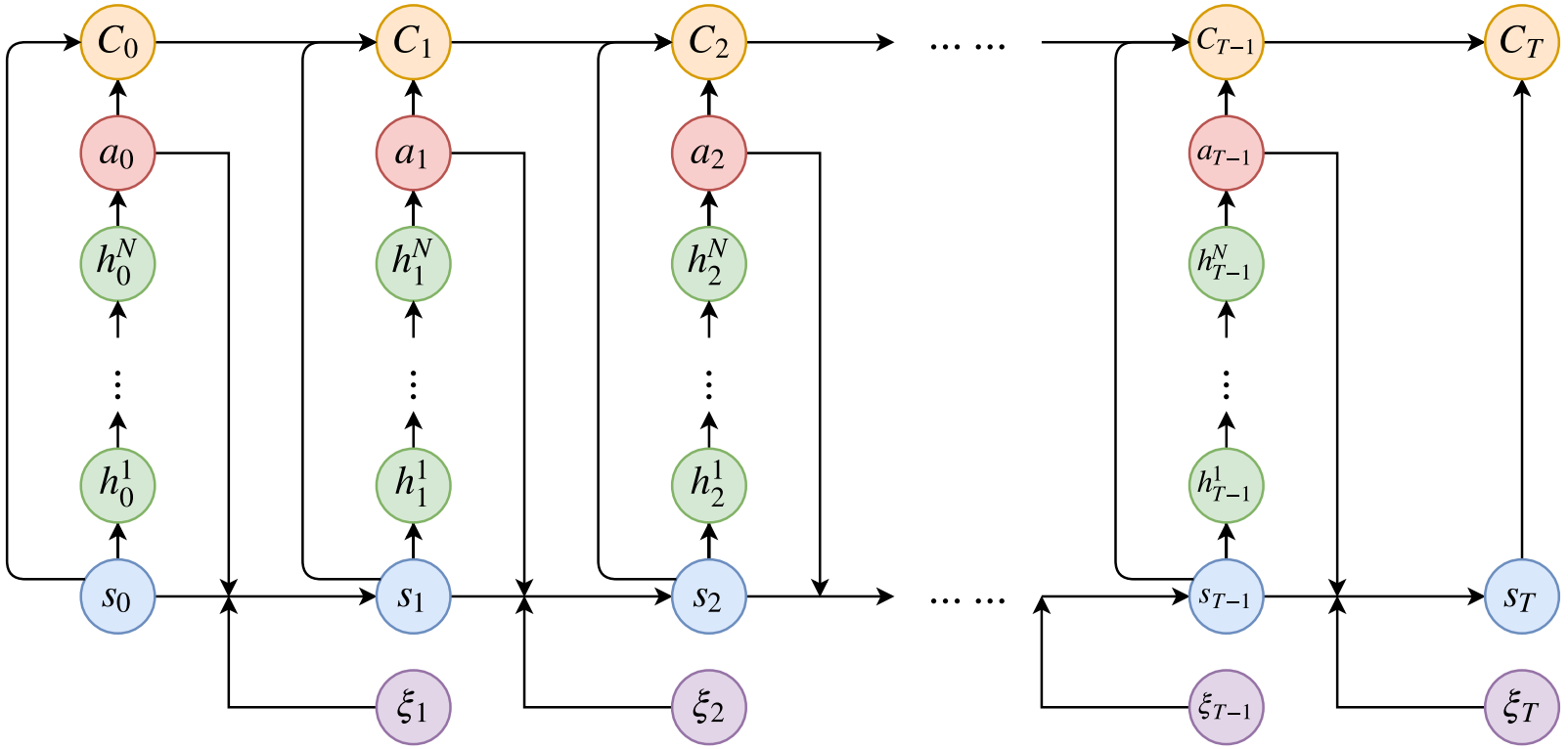


Figure: Network architecture for solving stochastic control in discrete time. The whole network has $(N + 1)T$ layers in total that involve free parameters to be optimized simultaneously. Each column (except ξ_t) corresponds to a sub-network at t.

Example: Energy Storage with Multiple Devices

The setting is similar to the above but now there are multiple devices, in which we do not find any other available solution for comparison.

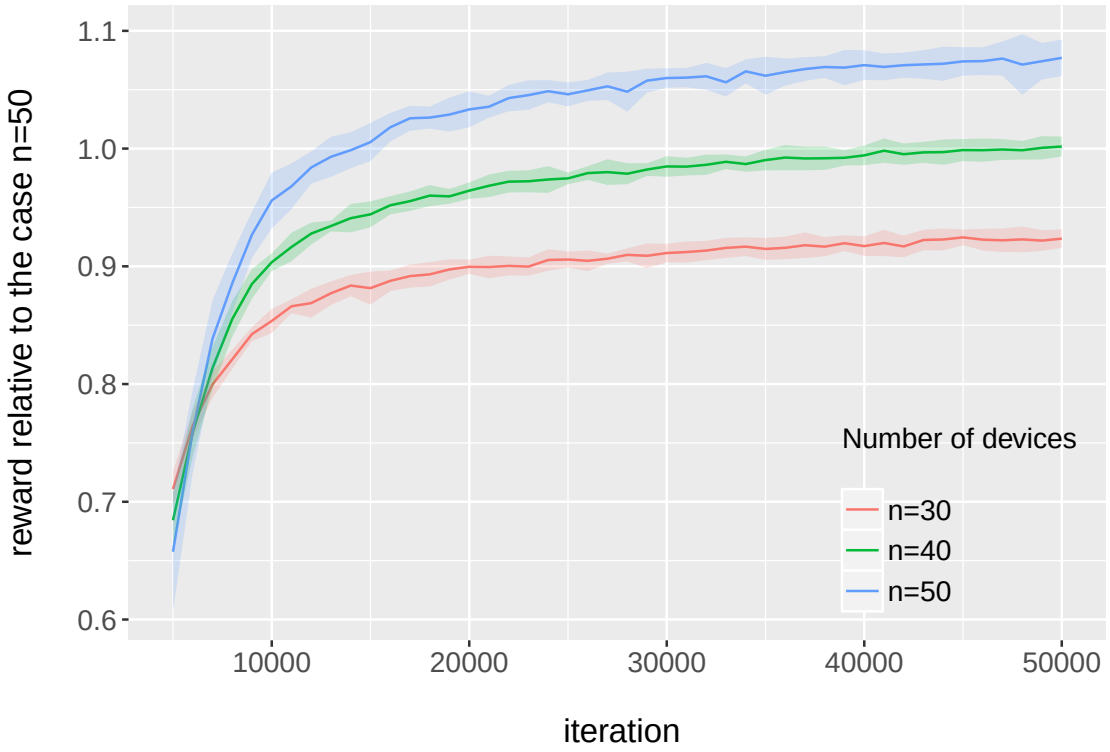


Figure: Relative reward. The space of control function is $\mathbb{R}^{n+2} \to \mathbb{R}^{3n}$ for $n = 30, 40, 50$, with multiple equality and inequality constrains.

5. Nonlinear parabolic PDE

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

Reformulating as a stochastic optimization problem using backward stochastic differential equations (BSDE, Pardoux and Peng (1990))

$$\begin{aligned} &\inf_{Y_0, \{Z_t\}_{0 \le t \le T}} \mathbb{E}|g(X_T) - Y_T|^2, \\ &s.t. \quad X_t = \xi + \int_0^t \mu(s, X_s)\, ds + \int_0^t \Sigma(s, X_s)\, dW_s, \\ &\qquad Y_t = Y_0 - \int_0^t h(s, X_s, Y_s, Z_s)\, ds + \int_0^t (Z_s)^{\mathrm{T}}\, dW_s. \end{aligned}$$

The unique minimizer is the solution to the PDE with:

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

Deep BSDE Method

- 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.
- This network 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 of given terminal condition defines the expected loss function

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

E, Han and Jentzen (Comm Math Stats 2017); Han, Jentzen and E (PNAS 2018)

Stochastic control revisited

LQG (linear quadratic Gaussian) for d=100

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

Cost functional: $J(\{m_t\}_{0\le t\le T}) = \mathbb{E}\big[\int_0^T \|m_t\|_2^2\, dt + g(X_T)\big]$.

HJB equation:

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

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

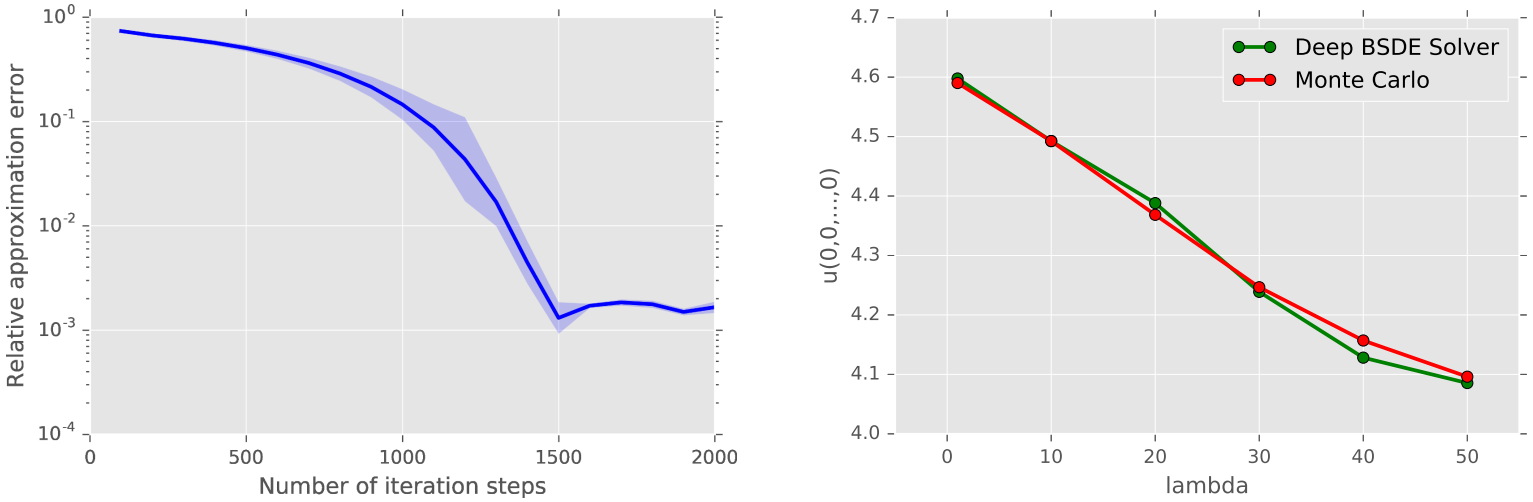


Figure: Left: Relative error of the deep BSDE method for $u(t{=}0, x{=}(0,\ldots,0))$ when $\lambda = 1$, which achieves 0.17% in a runtime of 330 seconds. Right: Optimal cost $u(t{=}0, x{=}(0,\ldots,0))$ against different λ.

Black-Scholes Equation with Default Risk

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

Q is some nonlinear function (Duffie et al. 1996, Bender et al. 2015 ($d = 5$))

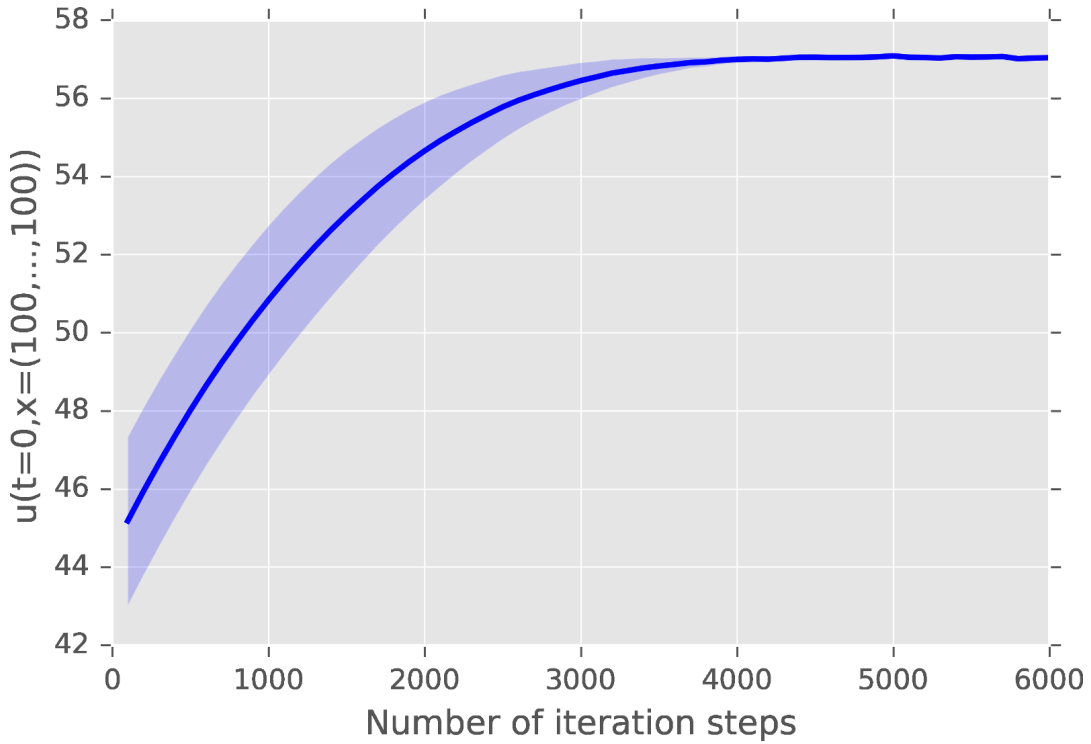
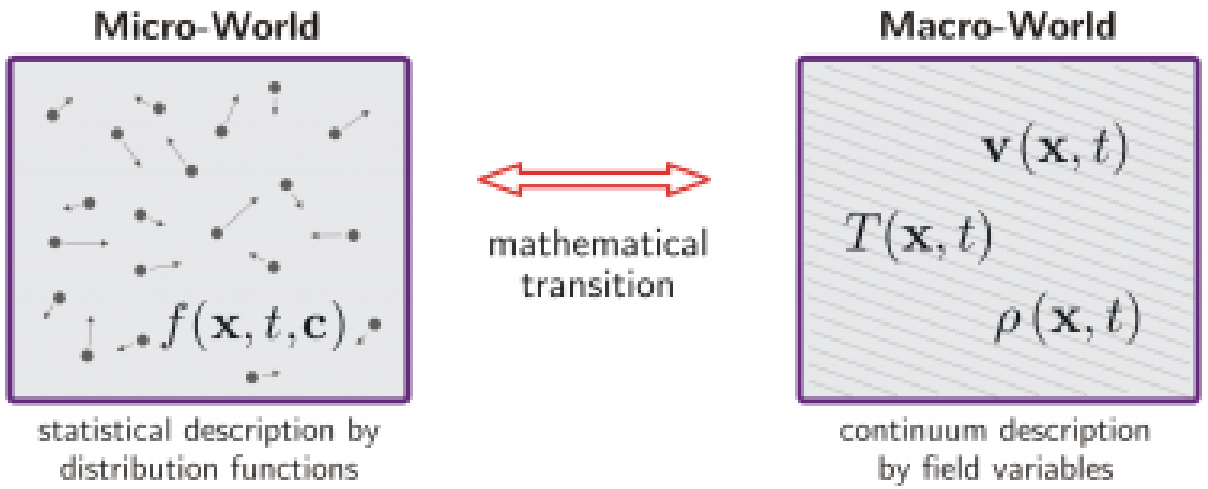


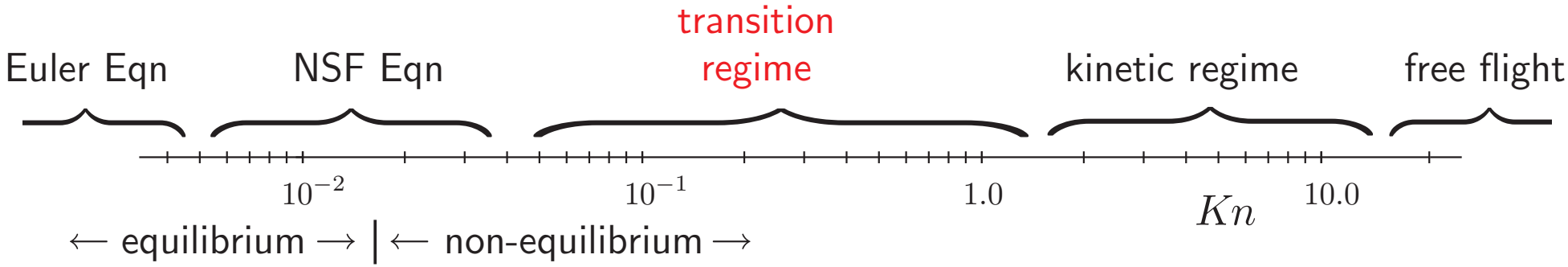
Figure: $d = 100$. Deep BSDE achieves a relative error of size 0.46% in a runtime of 617 seconds.

Applications to pricing basket options, interest rate-dependent options, Libor market model, Bermudan Swaption, barrier option, etc.

6. Modeling gas dynamics

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





Boltzmann Equation

$f(\boldsymbol{x}, \boldsymbol{v}, t) =$ phase space density function, $\varepsilon = Kn$, $Q =$ the collision operator

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

When $\varepsilon \ll 1$, this can be approximated by Euler (projection of Boltzmann on low order moments):

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

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

When ε is not small, seek generalization of Euler using more moments.

Grad 13-moment system is constructed using the moments of $\{1, \boldsymbol{v}, (\boldsymbol{v} - \boldsymbol{u}) \otimes (\boldsymbol{v} - \boldsymbol{u}), |\boldsymbol{v} - \boldsymbol{u}|^2 (\boldsymbol{v} - \boldsymbol{u})\}$.

Machine learning-based moment method

Objective: construct an uniformly accurate (generalized) moment model

1: Learn the Moments through Autoencoder
Find an encoder Ψ and a decoder Φ that recovers the original f from $\boldsymbol{U}, \boldsymbol{W}$

$$\boldsymbol{W} = \Psi(f) = \int \boldsymbol{w} f \, \mathrm{d}\boldsymbol{v}, \quad \Phi(\boldsymbol{U}, \boldsymbol{W})(\boldsymbol{v}) = \boldsymbol{h}(\boldsymbol{v}; \boldsymbol{U}, \boldsymbol{W}).$$

$$\text{Minimize}_{\boldsymbol{w},\boldsymbol{h}} \mathop{\mathbb{E}}_{f\sim\mathcal{D}} \|f - \Phi(\Psi(f))\|^2.$$

2: Learn the Fluxes and Source Terms in the PDE

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

We just have to learn $\boldsymbol{F}, \boldsymbol{G}, \boldsymbol{R}$ from the original kinetic equation using ML.

Jiequn Han, Chao Ma, Zheng Ma and Weinan E, PNAS. (2019)

ε varies from 10^{-3} to 10 in the domain; initial profiles are the same as before

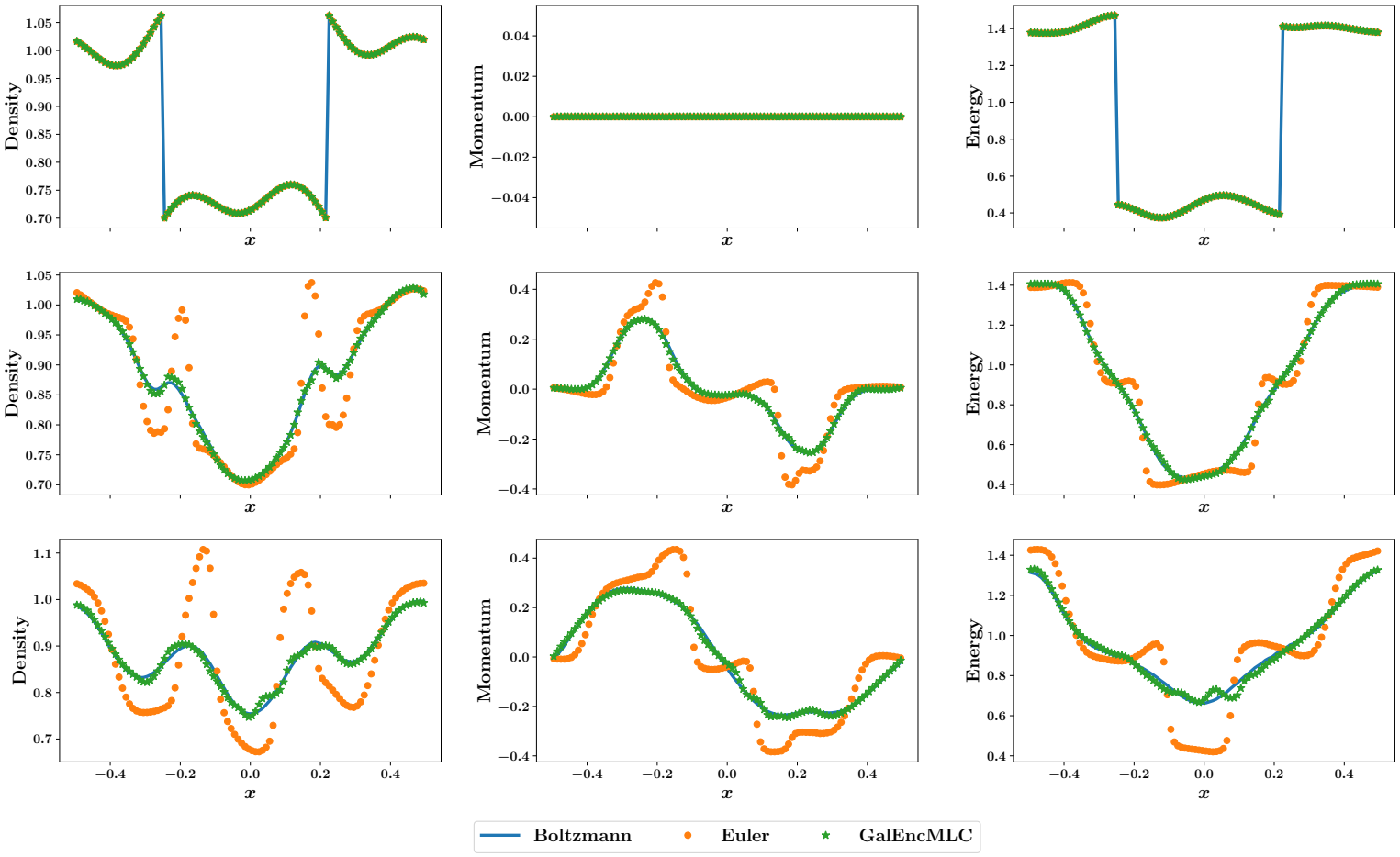


Figure: Profiles of $\rho, \rho\boldsymbol{u}, E$ (from left to right) at $t = 0, 0.05, 0.1$ (from top to bottom)

Follow-up work

- DeePHF, DeePKS, DeePN^2, CGSP, DeePCombustion
- deterministic control (Kang, Gong, et al (2019))
- game theory (Han and Hu (2019), Ruthotto, Osher et al (2020))
- Deep Ritz method (E and Yu (2018))
  "deep Galerkin method" (really least square, Sirignano and Spiliopoulos (2018))
  deep Galerkin method (Zang, Bao (2019))
- many applications to chemistry, material science, combustion, non-Newtonian fluid dynamics, control theory, finance, economics

ML is used to generate new (reliable and interpretable) physical models (say for gas dynamics, non-Newtonian fluids).

See E, Han and Zhang: *Integrating ML with Physics-based Modeling*, 2020.

Other examples

- many-body Schrödinger equation
- parametric PDEs
- solving traditional low-dimensional PDEs (Poisson, Maxwell, Navier-Stokes)
- inverse problems
- . . . . . .

1. Understanding the mysteries about ML

- Why does it work in such high dim?
- Why simple gradient descent works?
- Relative merits of shallow vs deep networks?
- Is over-parametrization good or bad? (cause the optimization problem to be degenerate)
- Why does neural network modeling require such extensive parameter tuning?

2. Seeking better formulations of ML

- More robust: requires less parameter tuning
- More general

Will discuss supervised learning: Approximate a target function using a finite dataset

Approximation of functions

Approximate by polynomials:

- "Universal Approximation Theorem" (Weierstrass): Continuous functions can be approximated by polynomials.
- Taylor's theorem: Convergence rates depend on the regularity of the target function.

Approximation by piecewise polynomials: (m = number of free parameters)

$$\inf_{f \in \mathcal{H}_m} \|f - f_m\|_{L^2(X)} \leq C_0 h^\alpha \|f\|_{H^\alpha(X)}, \qquad h \sim m^{-1/d}$$

- Sobolev (Besov) norm is the right quantity for the right hand side.
- They suffer from CoD: $m \sim \epsilon^{-d}$ where ϵ is the error tolerance.

The number of monomials of degree p in dimension d is C^d_{p+d}

What should we expect in high dimension?

Example: Monte Carlo methods for integration

$$I(g) = \mathbb{E}_{\boldsymbol{x}\sim\mu} g(\boldsymbol{x}), \quad I_m(g) = \frac{1}{m}\sum_j g(\boldsymbol{x}_j)$$

$\{\boldsymbol{x}_j, j \in [m]\}$ is i.i.d samples of μ.

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

The best we can expect for function approximation in high D:

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

- What should be the norm $\|\cdot\|_*$ (associated with the choice of $\mathcal{H}_m$)?

How can this be true? An illustrative example

Traditional approach for Fourier transform:

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

$\{\boldsymbol{\omega}_j\}$ is a fixed grid, e.g. uniform.

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

"New" approach: Let π be a probability distribution and

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

Let $\{\boldsymbol{\omega}_j\}$ be an i.i.d. sample of π, $f_m(\boldsymbol{x}) = \frac{1}{m} \sum_{j=1}^m a(\boldsymbol{\omega}_j) e^{i(\boldsymbol{\omega}_j,\boldsymbol{x})}$,

$$\mathbb{E}|f(\boldsymbol{x}) - f_m(\boldsymbol{x})|^2 = m^{-1} \mathsf{var}(f)$$

$f_m(\boldsymbol{x}) = \frac{1}{m} \sum_{j=1}^m a_j \sigma(\boldsymbol{\omega}_j^T \boldsymbol{x}) =$ two-layer neural network with activation function $\sigma(z) = e^{iz}$.

Two-layer neural network model: Barron spaces

E, Ma and Wu (2018, 2019), Bach (2017)

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

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

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

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

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

where $P_f := \{\rho : f(\boldsymbol{x}) = \mathbb{E}_\rho[a\sigma(\boldsymbol{w}^T \boldsymbol{x})]\}$.

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

Related work in Barron (1993), Klusowski and Barron (2016), E and Wojtowytsch (2020)

Theorem (Direct Approximation Theorem)

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

Theorem (Inverse Approximation Theorem)

Let

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

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 $\|f^*\|_{\mathcal{B}} \leq C$.

Estimation error

Since we can only work with a finite dataset, what happens outside the dataset?

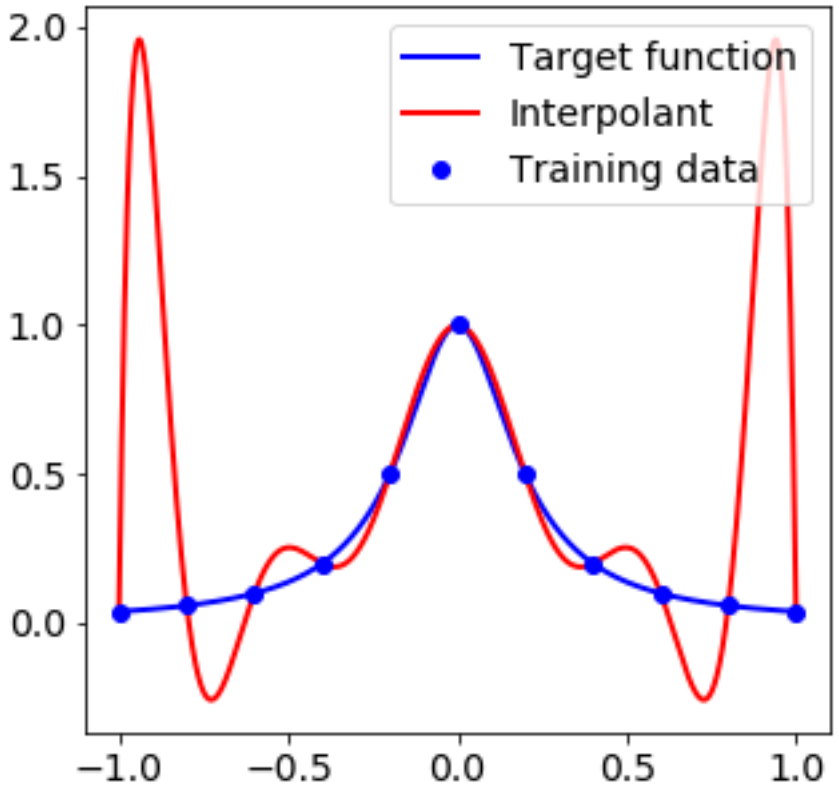


Figure: The Runge phenomenon: $f^*(x) = \frac{1}{1+25x^2}$

Training and testing errors

In practice, we 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:

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

$\mathcal{H}$ = a set of functions, $S = (\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_n)$ = dataset. Upto log terms,

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

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

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

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

Complexity estimates

Theorem (Bach, 2017)

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

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

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

A priori estimates for regularized model

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

where the path norm is defined by:

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

Theorem (E, Ma, Wu, 2018)

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

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

Approximation theory and function spaces for other ML models

- random feature model: Reproducing kernel Hilbert space (RKHS)
- Residual networks (ResNets): Flow-induced space (E, Ma and Wu (2019))
- Multi-layer neural networks: Multi-layer spaces (E and Wojtowytsch (2020))

Up to log terms, we have

$$\mathcal{R}(\hat{f}) \lesssim \frac{\|f^*\|_*^2}{m} + \frac{\|f^*\|_*}{\sqrt{n}}$$

where $m =$ number of free parameters, $n =$ size of training dataset.

Better formulation: ML from a continuous viewpoint

Formulate a "nice" continuous problem, then discretize to get concrete models/algorithms.

- For PDEs, "nice" = well-posed.
- For calculus of variation problems, "nice" = "convex", lower semi-continuous.
- For ML, "nice" = variational problem has simple landscape.

Key ingredients

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

E, Ma and Wu (2019)

Function representation

- integral-transform based:

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

  $\theta =$ parameters in the model: $a(\cdot)$ or the prob distributions π or ρ

- flow-based:

$$\begin{aligned} \frac{d\boldsymbol{z}}{d\tau} =& \mathbb{E}_{\boldsymbol{w}\sim\pi_\tau} \boldsymbol{a}(\boldsymbol{w},\tau)\sigma(\boldsymbol{w}^T\boldsymbol{z}) \\ =& \mathbb{E}_{(\boldsymbol{a},\boldsymbol{w})\sim\rho_\tau} \boldsymbol{a}\sigma(\boldsymbol{w}^T\boldsymbol{z}) \\ =& \mathbb{E}_{\boldsymbol{u}\sim\rho_\tau}\phi(\boldsymbol{z},\boldsymbol{u}), \quad \boldsymbol{z}(0,\boldsymbol{x}) = \boldsymbol{x} \end{aligned}$$

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

$\theta = \{a_\tau(\cdot)\}$ or $\{\pi_\tau\}$ or $\{\rho_\tau\}$

Optimization: Gradient flows

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

$$f(\boldsymbol{x}) = \int a(\boldsymbol{w})\sigma(\boldsymbol{w}^T\boldsymbol{x})\pi(d\boldsymbol{w}) = \mathbb{E}_{\boldsymbol{w}\sim\pi}a(\boldsymbol{w})\sigma(\boldsymbol{w}^T\boldsymbol{x})$$

Follow Halperin and Hohenberg (1977):

- $a =$ non-conserved, use "model A" dynamics:

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

- $\pi =$ conserved (probability density), use "model B":

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

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

Discretizing the gradient flows

- Discretizing the population risk (into the empirical risk) using data
- Discretizing the gradient flow
  - particle method – the dynamic version of Monte Carlo
  - smoothed particle method – analog of vortex blob method
  - spectral method – very effective in low dimensions

We can see that gradient descent algorithm (GD) for random feature and neural network models are simply the particle method discretization of the gradient flows discussed before.

Discretization of the conservative flow for flow-induced representation

Function representation: $f(\boldsymbol{x};\theta) = \mathbb{E}_{(a,\boldsymbol{w})\sim\rho} a\sigma(\boldsymbol{w}^T\boldsymbol{x})$

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

Particle method discretization:

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

gives rise 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.

Why is continuous formulation better? No "phase transition"

Continuous viewpoint (in this case same as mean-field): $f_m(\boldsymbol{x}) = \frac{1}{m}\sum_j a_j \sigma(\boldsymbol{w}_j^T \boldsymbol{x})$

Conventional NN models: $f_m(\boldsymbol{x}) = \sum_j a_j \sigma(\boldsymbol{w}_j^T \boldsymbol{x})$

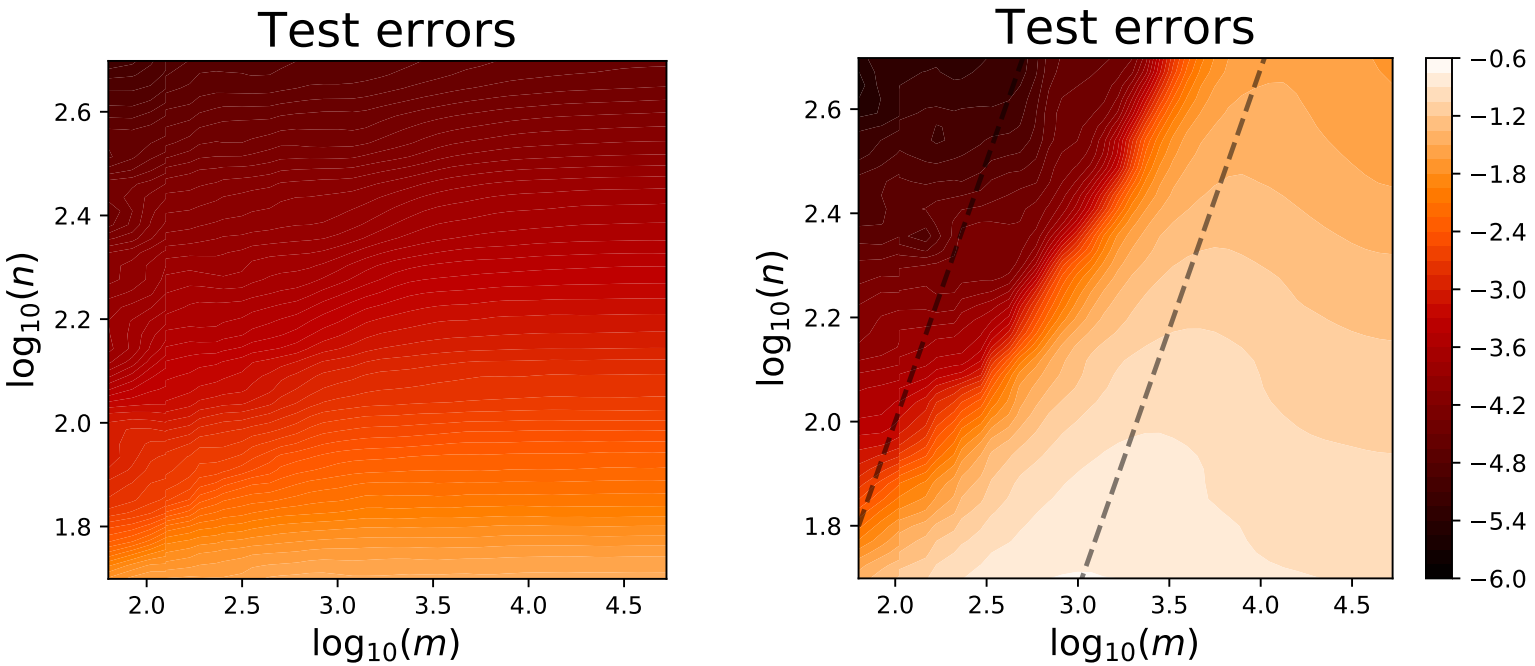


Figure: **(Left)** continuous viewpoint; **(Right)** conventional NN models. Target function is a single neuron.

Ma, Wu and E (2020)

The optimal control problem for flow-induced formulation

In a slightly more general form

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

$\boldsymbol{z}$ = state, ρ_τ = control at time τ.

The objective : 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

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

Pontryagin's maximum principle

Define the Hamiltonian $H : \mathbb{R}^d \times \mathbb{R}^d \times \mathcal{P}_2(\Omega) :\mapsto \mathbb{R}$ as

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

The solutions of the control problem must satisfy:

$$\rho_\tau = \text{argmax}_\rho \, \mathbb{E}_{\boldsymbol{x}}[H\left(\boldsymbol{z}_\tau^{t,\boldsymbol{x}}, \boldsymbol{p}_\tau^{t,\boldsymbol{x}}, \rho\right)], \;\; \forall \tau \in [0, 1],$$

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

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

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

with the boundary conditions:

$$\boldsymbol{z}_0^{t,\boldsymbol{x}} = \boldsymbol{x}$$
$$\boldsymbol{p}_1^{t,\boldsymbol{x}} = 2(f(\boldsymbol{x}; \rho(\cdot; t)) - f^*(\boldsymbol{x}))\mathbf{1}.$$

Gradient flow for flow-based models

Define the Hamiltonian $H : \mathbb{R}^d \times \mathbb{R}^d \times \mathcal{P}_2(\Omega) :\mapsto \mathbb{R}$ as

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

The gradient flow for $\{\rho_\tau\}$ is given by

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

where

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

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

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

with the boundary conditions:

$$\boldsymbol{z}_0^{t,\boldsymbol{x}} = \boldsymbol{x}$$
$$\boldsymbol{p}_1^{t,\boldsymbol{x}} = 2(f(\boldsymbol{x}; \rho(\cdot; t)) - f^*(\boldsymbol{x}))\mathbf{1}.$$

Discretize the gradient flow

- forward Euler for the flow in τ variable, step size $1/L$.
- particle method for the GD dynamics, M samples in each layer

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

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

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

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

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

Max principle-based training algorithm

Qianxiao Li, Long Chen, Cheng Tai and Weinan E (2017):

Basic "method of successive approximation" (MSA):

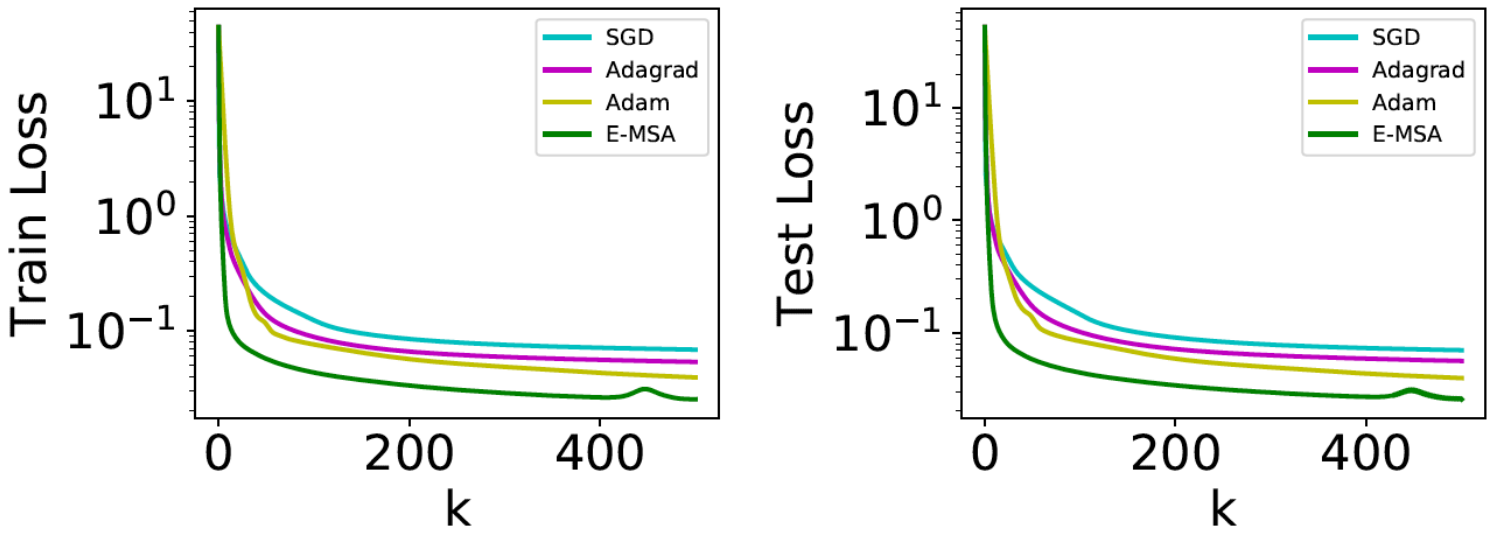
Initialize: $\theta^0 \in \mathcal{U}$

For $k = 0, 1, 2, \cdots$:

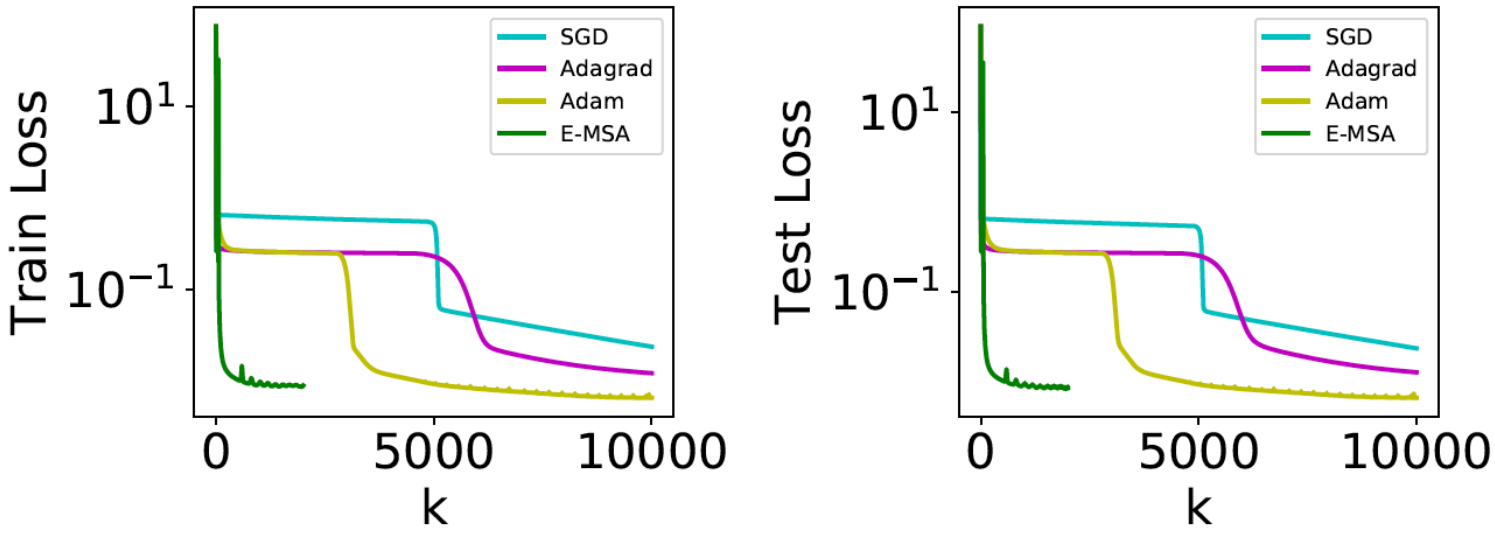
- Solve
$$\frac{d\boldsymbol{z}_\tau^k}{d\tau} = \nabla_{\boldsymbol{p}} H(\boldsymbol{z}_\tau^k, \boldsymbol{p}_\tau^k, \theta_\tau^k), \quad \boldsymbol{z}_0^k = V\boldsymbol{x}$$
- Solve
$$\frac{d\boldsymbol{p}_\tau^k}{d\tau} = -\nabla_{\boldsymbol{z}} H(\boldsymbol{z}_\tau^k, \boldsymbol{p}_\tau^k,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

Extended MSA:

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


Train Loss
Test Loss
SGD
Adagrad
Adam
E-MSA
10^1
10^0
10^{-1}
0
200
400
k

(a)


Train Loss
Test Loss
SGD
Adagrad
Adam
E-MSA
10^1
10^{-1}
0
5000
10000
k

(b)

Comparison between GD and maximum principle

Maximum principle:

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

GD:

$$\partial_t \rho_\tau(\boldsymbol{u}, t) = \nabla \cdot \left(\rho_\tau(\boldsymbol{u}, t) \nabla \mathbb{E}_{\boldsymbol{x}}[\frac{\delta H}{\delta \rho}\left(\boldsymbol{z}_\tau^{t,\boldsymbol{x}}, \boldsymbol{p}_\tau^{t,\boldsymbol{x}}, \rho_\tau(\boldsymbol{u}; t)\right)] \right), \ \forall \tau \in [0,1],$$

Hybrid:

Introducing a different time scale for optimization step: One time forward/backward propagation every k steps of optimization.

- $k = 1$, usual GD or SGD
- $k = \infty$, maximum principle

What have we really learned from ML?

Representation of functions as expectations:

- integral-transform based:

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

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

- flow-based:

$$\begin{aligned}\frac{d\boldsymbol{z}}{d\tau} =& \mathbb{E}_{(\boldsymbol{a},\boldsymbol{w})\sim\rho_\tau}\boldsymbol{a}\sigma(\boldsymbol{w}^T\boldsymbol{z}), \quad \boldsymbol{z}(0,\boldsymbol{x}) = \boldsymbol{x}\\ f(\boldsymbol{x},\theta) =& \mathbf{1}^T\boldsymbol{z}(1,\boldsymbol{x})\end{aligned}$$

and then discretize using particle, spectral or other numerical methods.

Concluding remarks

- ML has changed and will continue to change the way we deal with functions, and this will have a very significant impact in computational mathematics.
- A reasonable mathematical picture for ML is emerging, from the perspective of **numerical analysis**.

Review articles (can be found on my webpage *https://web.math.princeton.edu/ weinan*):

- Towards a mathematical understanding of machine learning: What is known and what is not (will appear soon)
- Algorithms for solving high dimensional PDEs: From nonlinear Monte Carlo to machine learning
- Integrating machine learning with physics-based modeling