

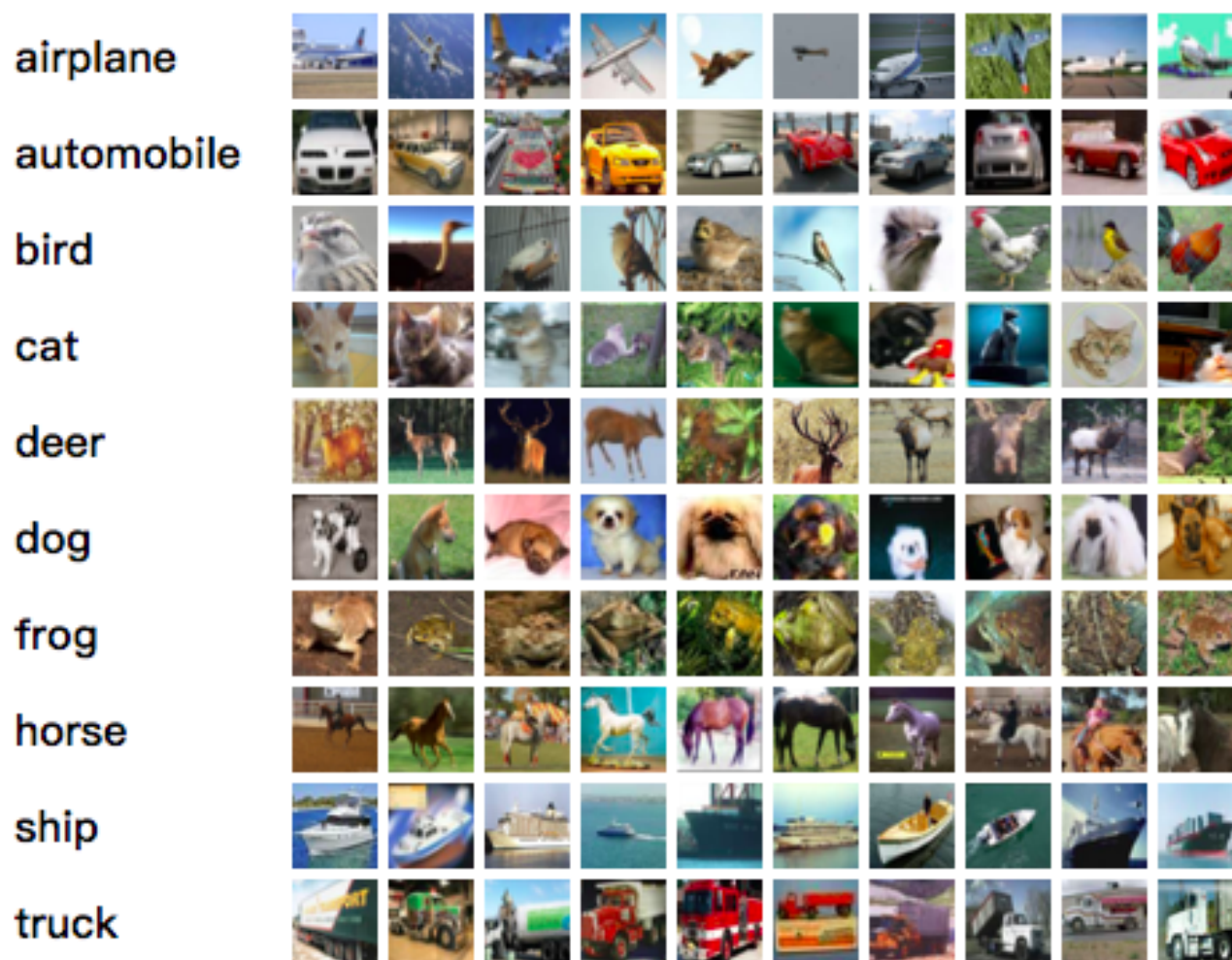
机器学习的数学原理

鄂维南

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: $f^* \in \{\text{airplane}, \dots, \text{truck}\}$.
- $f^* : [0, 1]^{3072} \rightarrow \{\text{airplane}, \dots, \text{truck}\}$.
 $f^*(\text{each image}) = \text{category}$

Sampling unknown high dimensional distributions

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



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.

The DNN - SGD paradigm

DNN = deep neural network, SGD = stochastic gradient descent

- 1 choose a hypothesis space (set of trial functions), e.g. 2-layer neural networks

$$f(\mathbf{x}, \theta) = \sum_{j=1}^m a_j \sigma(\mathbf{w}_j^T \mathbf{x}), \quad \theta = (a_j, \mathbf{w}_j)$$

σ is a nonlinear activation function, e.g. $\sigma(z) = \max(z, 0)$

- 2 choose a loss function (to fit the data), e.g. “empirical risk”

$$\hat{\mathcal{R}}_n(\theta) = \frac{1}{n} \sum_j (f(x_j, \theta) - f^*(x_j))^2 = \frac{1}{n} \sum_{j=1}^n \ell_j(\theta)$$

- 3 choose an optimization algorithm and parameters, e.g. gradient descent (GD)

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

Stochastic gradient descent (SGD):

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

j_1, j_2, \dots are iid random variables uniformly drawn from $1, 2, \dots, n$.

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

Model dynamics (analog of ResNet):

$$\mathbf{z}_{l+1} = \mathbf{z}_l + \mathbf{g}_l(\mathbf{z}_l, a_l) + \xi_l,$$

\mathbf{z}_l = state, a_l = control, ξ_l = noise.

$$\min_{\{a_l\}_{l=0}^{T-1}} \mathbb{E}_{\{\xi_l\}} \left\{ \sum_{l=0}^{T-1} c_l(\mathbf{z}_l, a_l(\mathbf{z}_l)) + c_T(\mathbf{z}_T) \right\},$$

Look for a feedback control:

$$a_l = a_l(\mathbf{z}).$$

- Neural network approximation:

$$a_l(\mathbf{z}) \approx \tilde{a}_l(\mathbf{z}|\theta_l), \quad l = 0, \dots, T-1$$

Optimization problem (SGD applies directly)

$$\min_{\{\theta_l\}_{l=0}^{T-1}} \mathbb{E}_{\{\xi_l\}} \left\{ \sum_{l=0}^{T-1} c_l(\mathbf{z}_l, \tilde{a}_l(\mathbf{z}_l|\theta_l)) + c_T(\mathbf{z}_T) \right\}$$

subject to $\mathbf{z}_{l+1} = \mathbf{z}_l + \mathbf{g}_l(\mathbf{z}_l, a_l) + \xi_l$.

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.

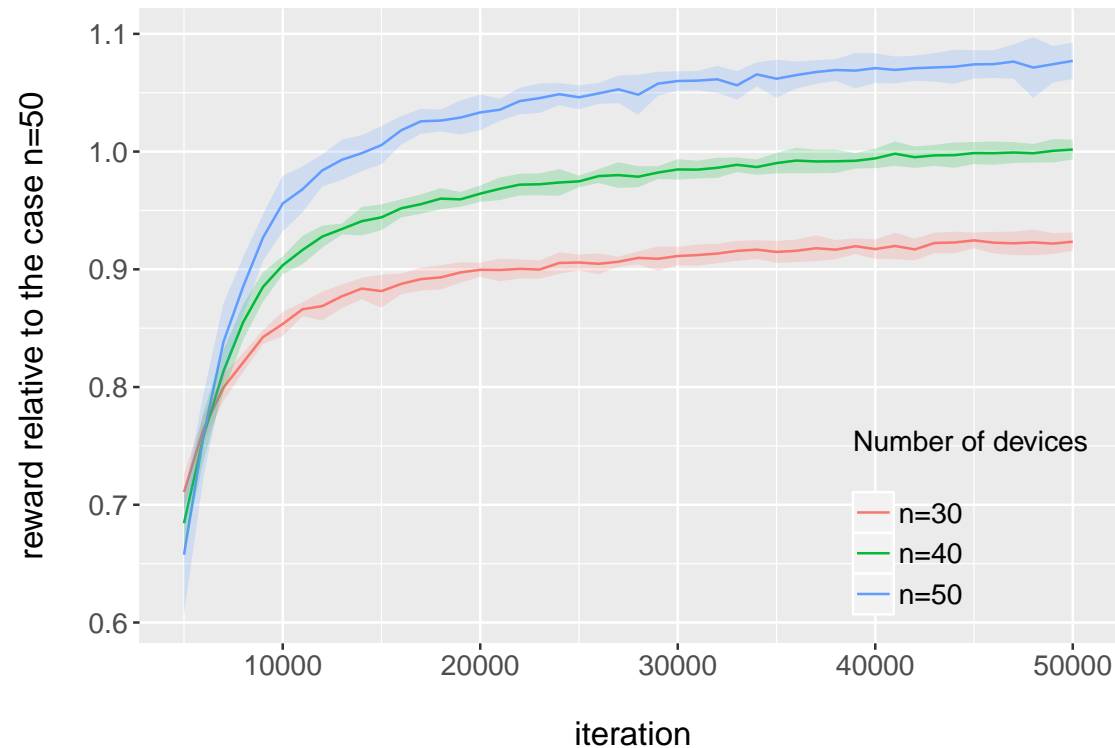


Figure: Relative reward. The space of control function is $\mathbb{R}^{n+2} \rightarrow \mathbb{R}^{3n}$ for $n = 30, 40, 50$, with multiple equality and inequality constraints.

2. Nonlinear parabolic PDE

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

or equivalently with $u(t, x) = v(T - t, x)$

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

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

$$\begin{aligned} & \inf_{Y_0, \{Z_t\}_{0 \leq t \leq T}} \mathbb{E} |g(X_0) - Y_0|^2, \\ \text{s.t. } & X_t = X_0 + \int_0^t \mu(s, X_s) ds + \int_0^t \sigma(s, X_s) dW_s, \\ & Y_t = Y_0 - \int_0^t f(Z_s) ds + \int_0^t (Z_s)^T dW_s. \end{aligned}$$

The unique minimizer is the solution to the PDE with:

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

Stochastic control revisited

LQG (linear quadratic Gaussian) for $d=100$

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

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

HJB equation:

$$\partial_t u + \Delta u - \lambda \|\nabla u\|_2^2 = 0$$

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

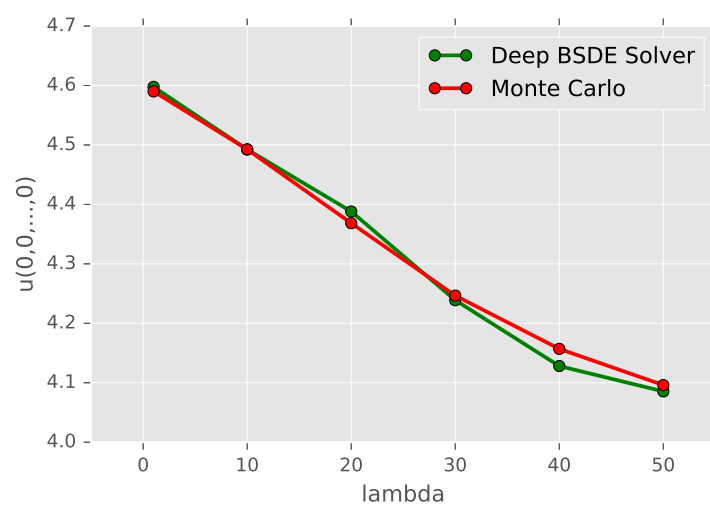
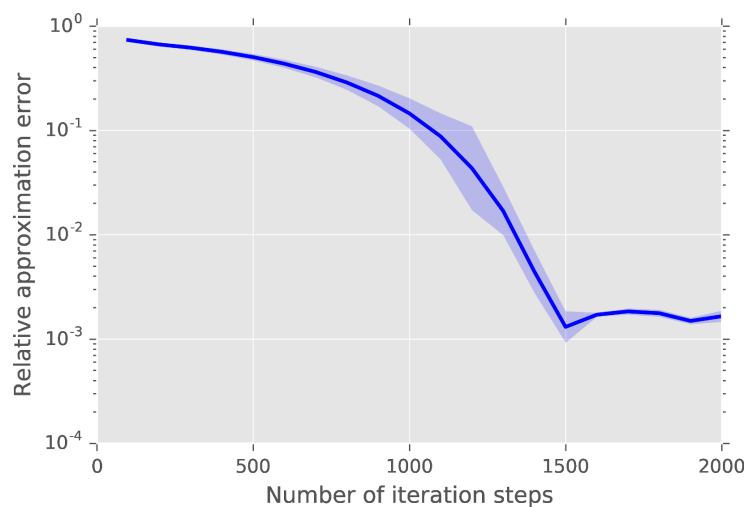










Figure: Left: Relative error of the deep BSDE method for $u(t=0, x=(0, \dots, 0))$ when $\lambda = 1$, which achieves 0.17% in a runtime of 330 seconds. Right: Optimal cost $u(t=0, x=(0, \dots, 0))$ against different λ .

Open source platform

 frankhan91 Small fix.	8ec1353 on Dec 21, 2019	🕒 6 commits
 configs	Updated version based on TF2.0.	10 months ago
 .gitignore	First version.	3 years ago
 LICENSE	Initial commit	3 years ago
 README.md	Updated version based on TF2.0.	10 months ago
 equation.py	Updated version based on TF2.0.	10 months ago
 main.py	Updated version based on TF2.0.	10 months ago
 solver.py	Small fix.	10 months ago

README.md

Deep BSDE Solver in TensorFlow (2.0)

Training

```
python main.py --config_path=configs/hjb_lq_d100.json
```

Deep BSDE solver in TensorFlow

[deep-learning](#)

[partial-differential-equations](#)

 Readme

 MIT License

Releases

No releases published

Packages

No packages published

Languages

● Python 100.0%

Jiequn Han et al.

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

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

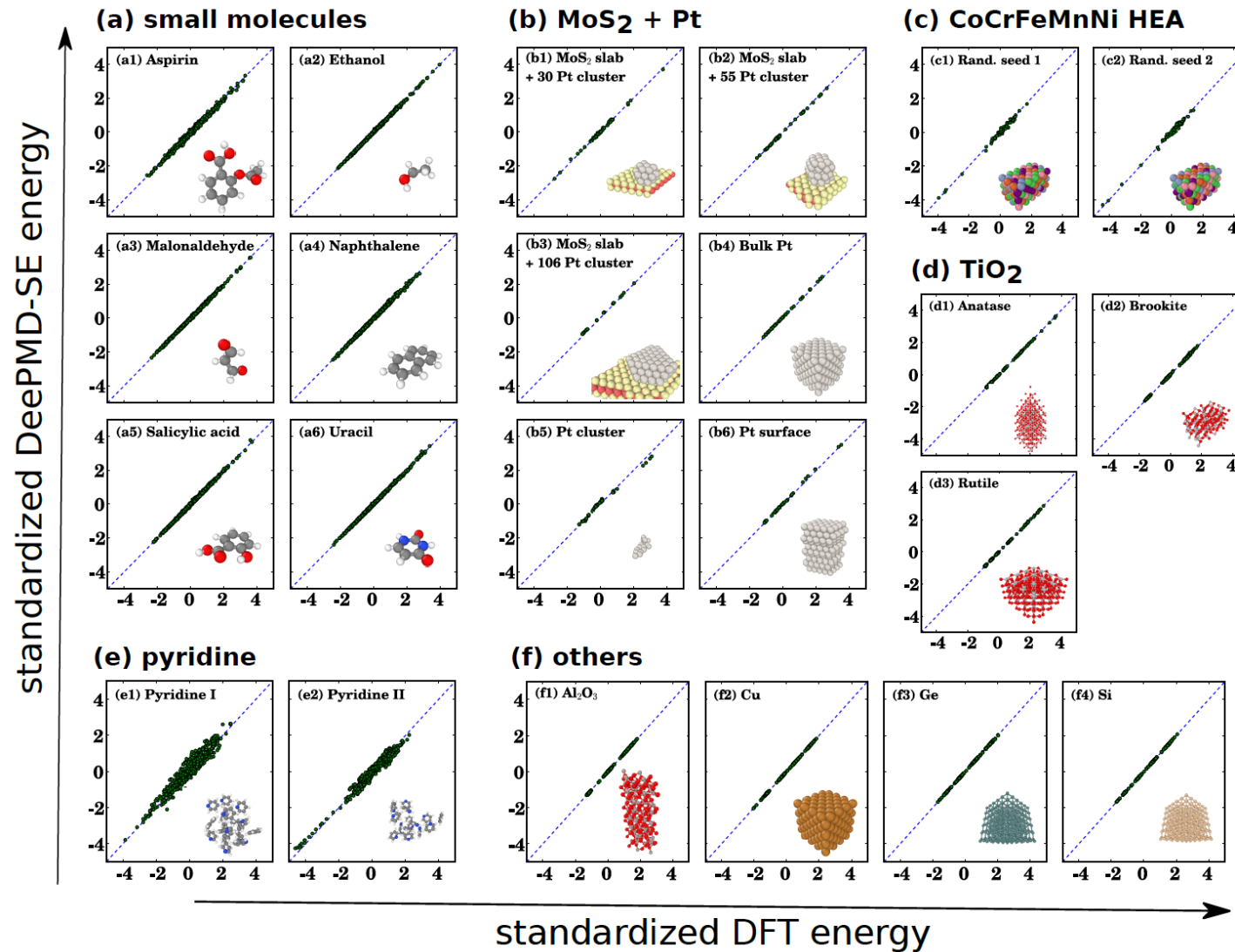
Key question: $V = ?$

Two ways to calculate V :

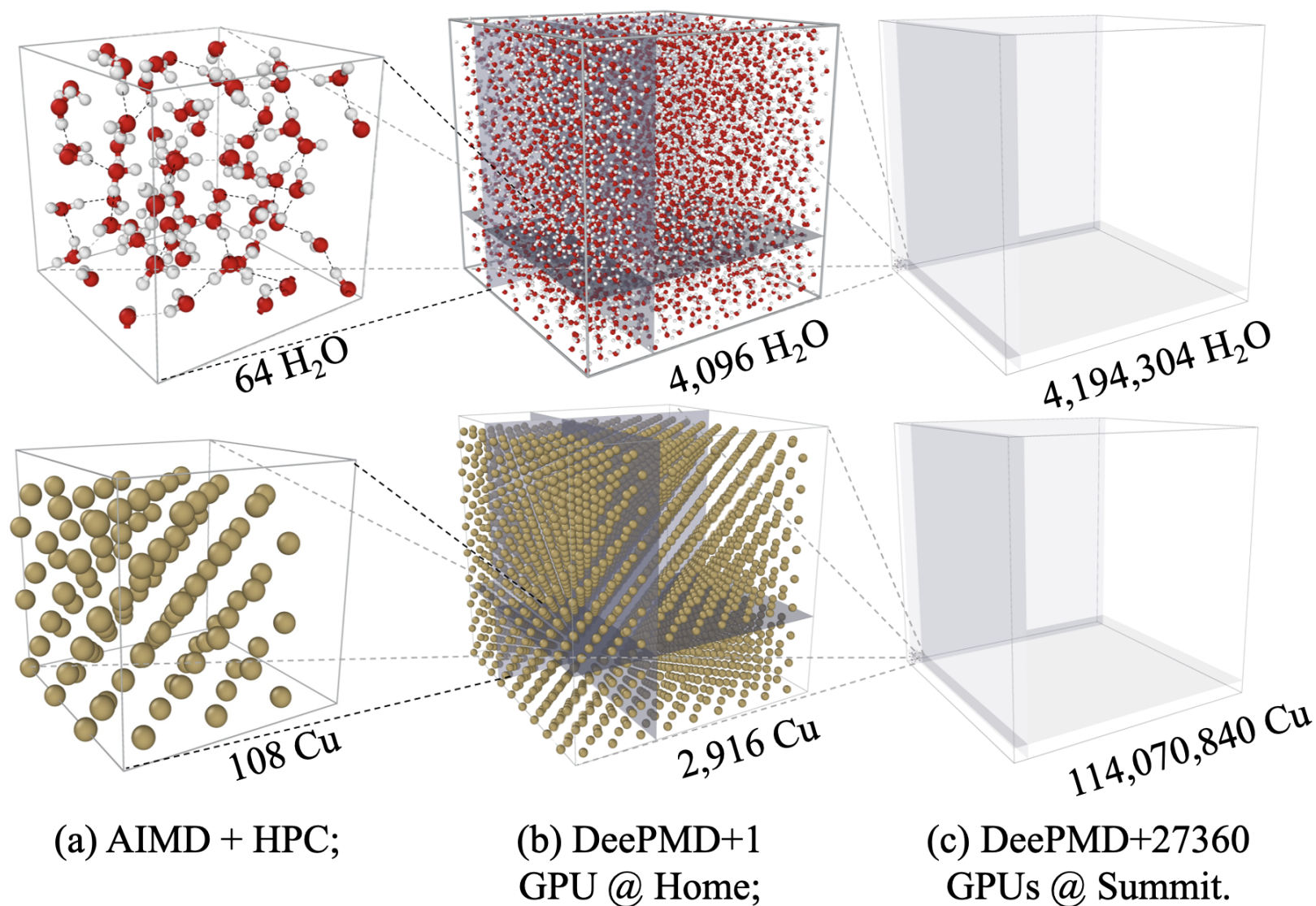
- Computing the inter-atomic forces on the fly using QM, e.g. the Car-Parrinello MD. Accurate but expensive (limited to about 1000 atoms).
- Empirical potentials: basically guess what V should be. Efficient but unreliable.

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



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





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


Open source platform

Type: All ▾

Language: All ▾

deepmd-kit
A deep learning package for many-body potential energy representation and molecular dynamics

C++ LGPL-3.0 99 265 66 3 Updated 2 days ago

dpdata
Manipulating DeePMD-kit, VASP, LAMMPS data formats.

Python LGPL-3.0 30 26 1 1 Updated 6 days ago

dpgen
The deep potential generator

Python LGPL-3.0 44 56 13 2 Updated 28 days ago

Top languages

Python C++

People

2 >



Linfeng Zhang, Jiequn Han, Han Wang et al.

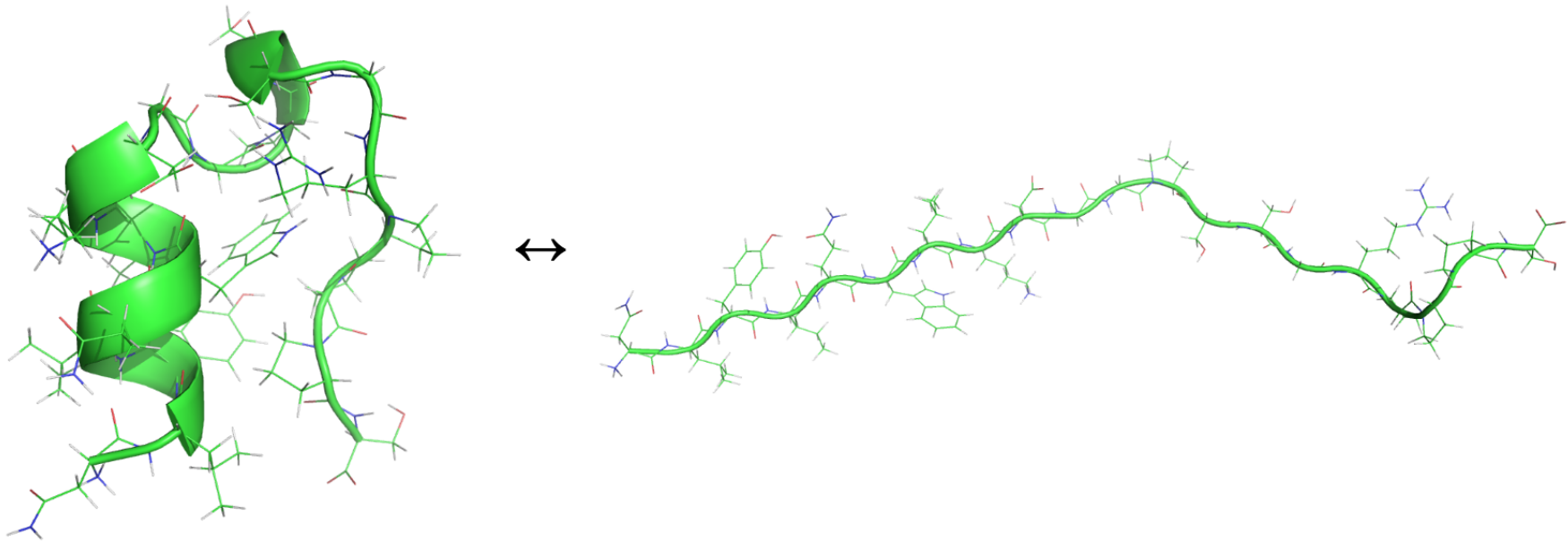
4. Protein folding and the nonlinear multi-grid method

$\{\mathbf{x}_j\}$ = positions of the atoms in a protein

$U(\{\mathbf{x}_j\})$ = potential energy (chemical bonding, Van der Waals, electro-static, etc).

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

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

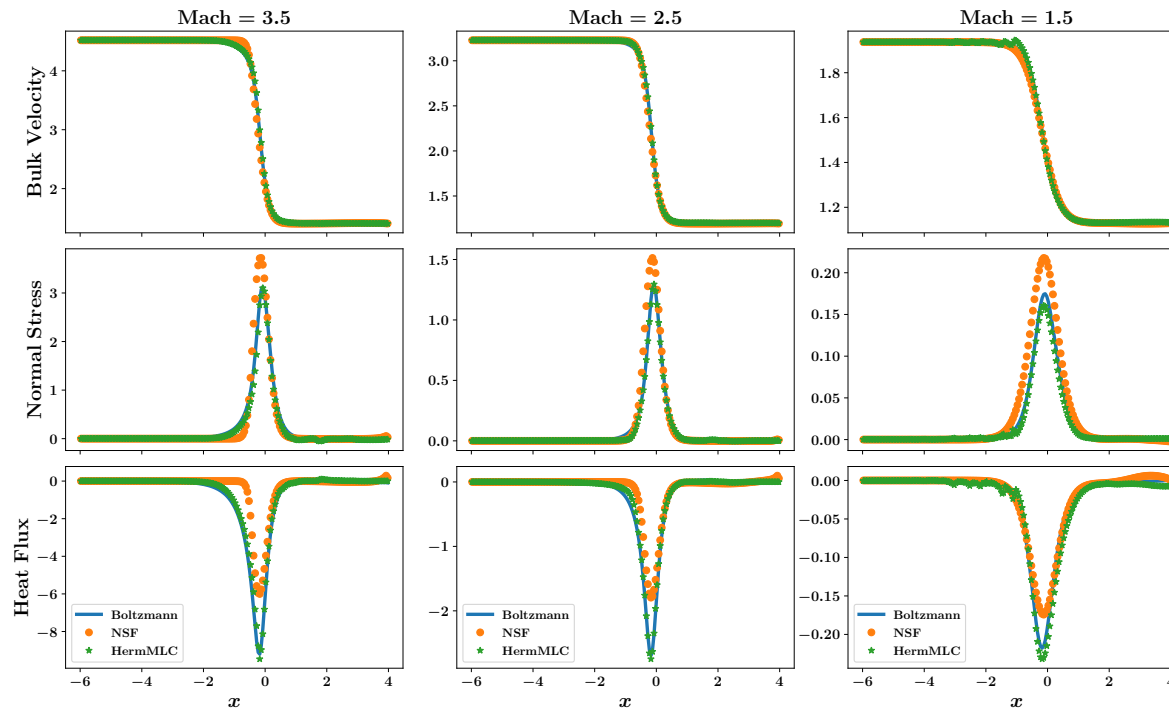


5. Moment closure for Boltzmann equation

$$f = f(\mathbf{x}, \mathbf{v}, t), \quad (\mathbf{U}, \mathbf{W}) = (\mathbf{U}, \mathbf{W})(\mathbf{x}, t) = \int g(\mathbf{v}) f(\mathbf{x}, \mathbf{v}, t) d\mathbf{v}$$

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

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



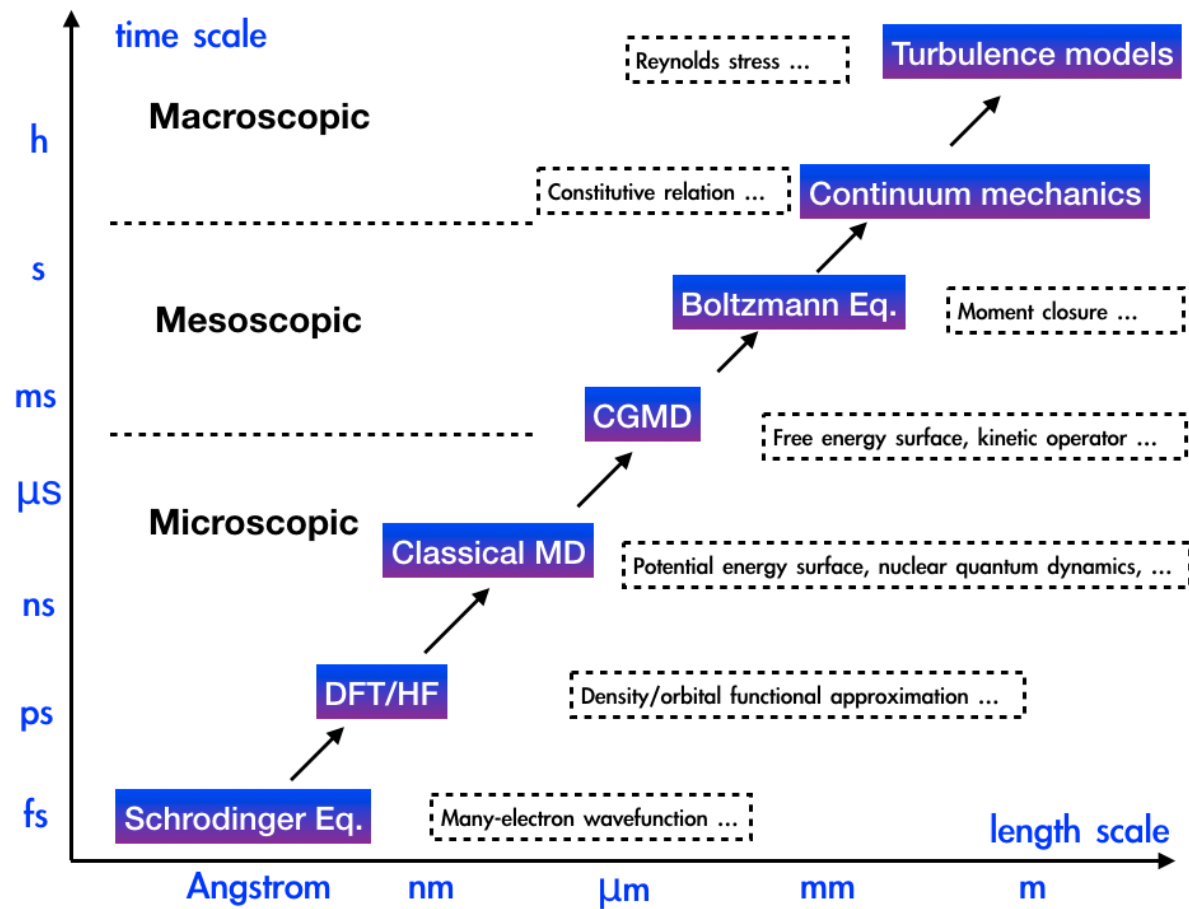


Figure: Representative physical models at different scale and their most important modeling ingredients.

- DeePHF, DeePKS
- Reinforced dynamics
- 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.

1. Understanding the mysteries about ML

- Why does it work in such high dim?
- Why is it so fragile? (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

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

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

- Sobolev (Besov) norm is the right quantity for the right hand side.
- They suffer from CoD: $m \sim \epsilon^{-d}$ (if $\alpha = 1$) where ϵ is the error tolerance.

What should we expect in high dimension?

Example: Monte Carlo methods for integration

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

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

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

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

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

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

Approximating functions in high D: An illustrative example

Traditional approach for Fourier transform:

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

$\{\boldsymbol{\omega}_j\}$ 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 $(\sigma(z) = e^{iz})$

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

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

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

$f_m(\mathbf{x}) = \frac{1}{m} \sum_{j=1}^m a_j \sigma(\boldsymbol{\omega}_j^T \mathbf{x}) = \text{two-layer neural network.}$

Two-layer neural network model: Barron spaces

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

$$\mathcal{H}_m = \{f_m(\mathbf{x}) = \frac{1}{m} \sum_j a_j \sigma(\mathbf{w}_j^T \mathbf{x})\}, \sigma(z) = \max(z, 0)$$

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

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

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

$$\|f\|_{\mathcal{B}} = \inf_{\rho: f(\mathbf{x}) = \mathbb{E}_{\rho}[a \sigma(\mathbf{w}^T \mathbf{x})]} \left(\mathbb{E}_{\rho}[a^2 \|\mathbf{w}\|_1^2] \right)^{1/2}$$

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

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

What kind of functions admit such a representation?

Theorem (Barron and Klusowski (2016)): If $\int_{\mathbb{R}^d} \|\boldsymbol{\omega}\|_1^2 |\hat{f}(\boldsymbol{\omega})| d\boldsymbol{\omega} < \infty$, where \hat{f} is the Fourier transform of f , then f can be represented as

$$\tilde{f}(\boldsymbol{x}) = f(\boldsymbol{x}) - (f(0) + \boldsymbol{x} \cdot \nabla f(0)) = \int_{\Omega} a \sigma(\boldsymbol{w}^T \boldsymbol{x}) \rho(da, d\boldsymbol{w})$$

for $\boldsymbol{x} \in [0, 1]^d$. Furthermore, we have

$$\mathbb{E}_{(a, \boldsymbol{w}) \sim \rho} |a| \|\boldsymbol{w}\|_1 \leq 2 \int_{\mathbb{R}^d} \|\boldsymbol{\omega}\|_1^2 |\hat{f}(\boldsymbol{\omega})| d\boldsymbol{\omega}$$

Theorem (Direct Approximation Theorem)

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

Theorem (Inverse Approximation Theorem)

Let

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

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

$$f_m(\mathbf{x}) \rightarrow f^*(\mathbf{x})$$

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

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

for all $\mathbf{x} \in X$ and $\|f^*\|_{\mathcal{B}} \leq C$.

Estimation error

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

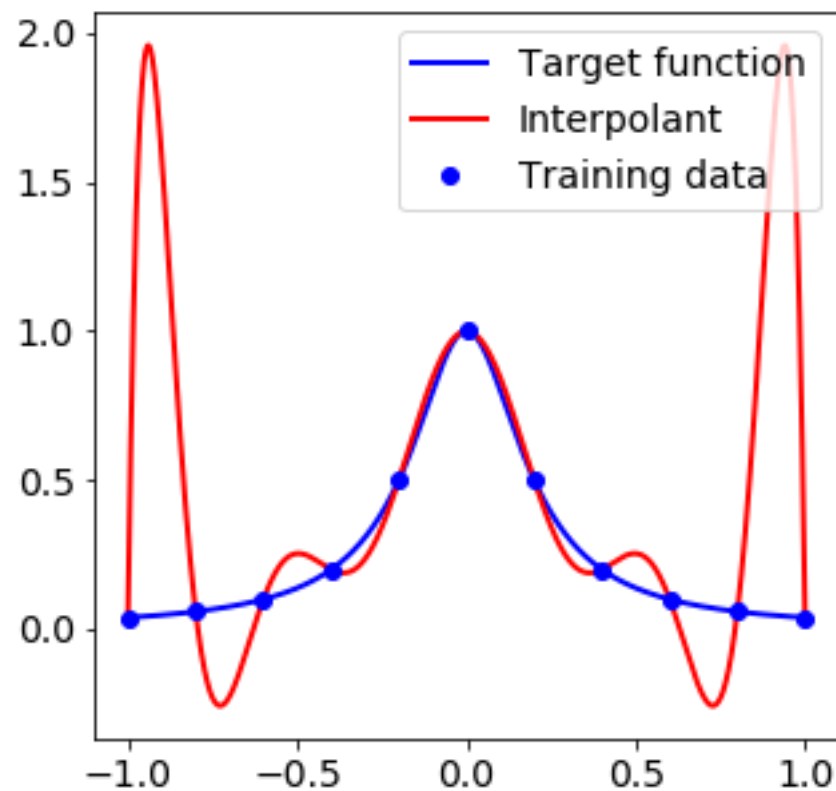


Figure: The Runge phenomenon: $f^*(x) = \frac{1}{1+25x^2}$

Training and testing errors

In practice, we minimize the training error:

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

but we are interested in the testing error:

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

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

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

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

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

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

Theorem (Bach, 2017)

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

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

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

A priori estimates for regularized model

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

where the path norm is defined by:

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

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:

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

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

- flow-based:

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

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

Optimization: Gradient flows

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

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

Follow Halperin and Hohenberg (1977):

- a = non-conserved, use “model A” dynamics:

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

- π = conserved (probability density), use “model B”:

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

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

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

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

Particle method discretization:

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

gives rise to

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

where

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

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): $f_m(\mathbf{x}) = \frac{1}{m} \sum_j a_j \sigma(\mathbf{w}_j^T \mathbf{x})$

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

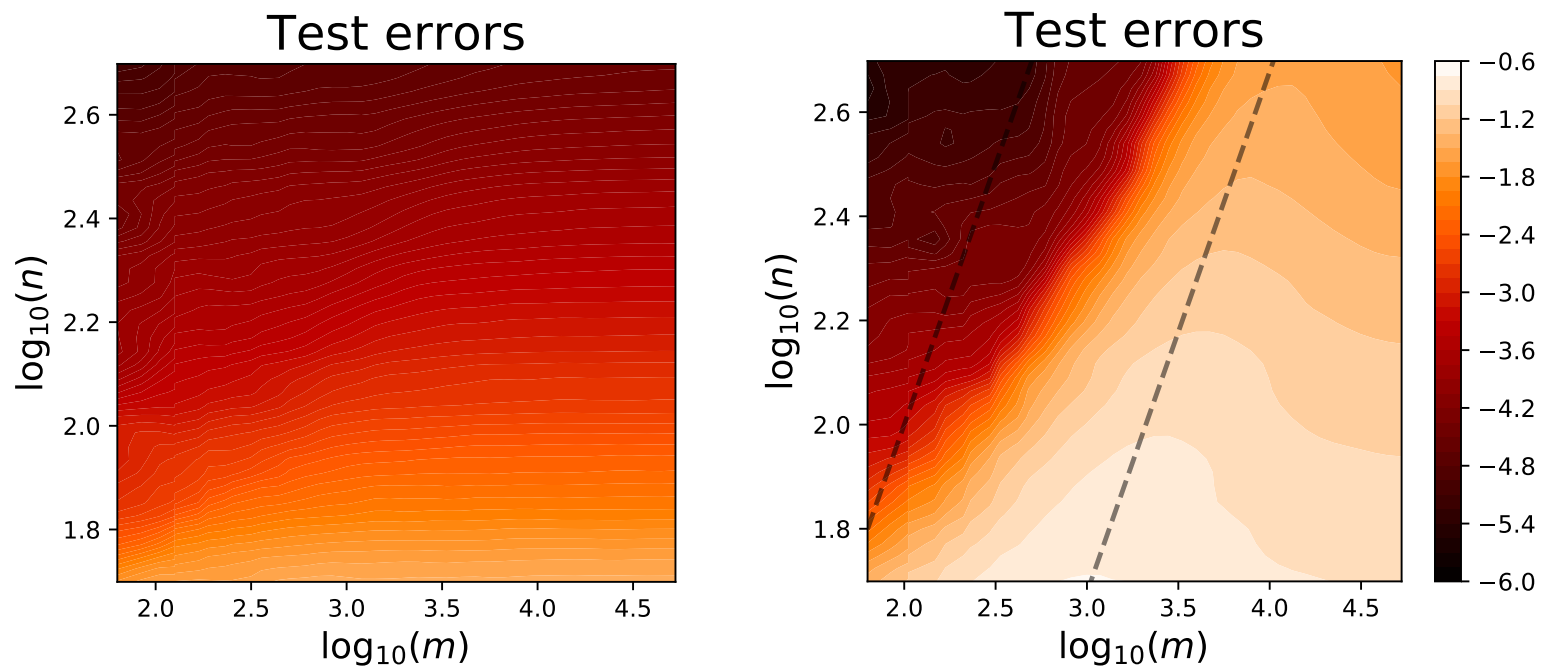


Figure: **(Left)** continuous viewpoint; **(Right)** conventional NN models. Target function is a single neuron.

Ma, Wu and E (2020)

The optimal control problem for flow-induced formulation

In a slightly more general form

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

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

The objective : Minimize \mathcal{R} over $\{\rho_\tau\}$

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

where

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

Pontryagin's maximum principle

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

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

The solutions of the control problem must satisfy:

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

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

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

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

with the boundary conditions:

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

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

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

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

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

where

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

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

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

with the boundary conditions:

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

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

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

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

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

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

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

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 $k = 0, 1, 2, \dots$:

- Solve

$$\frac{d\mathbf{z}_\tau^k}{d\tau} = \nabla_{\mathbf{p}} H(\mathbf{z}_\tau^k, \mathbf{p}_\tau^k, \theta_\tau^k), \quad \mathbf{z}_0^k = V\mathbf{x}$$

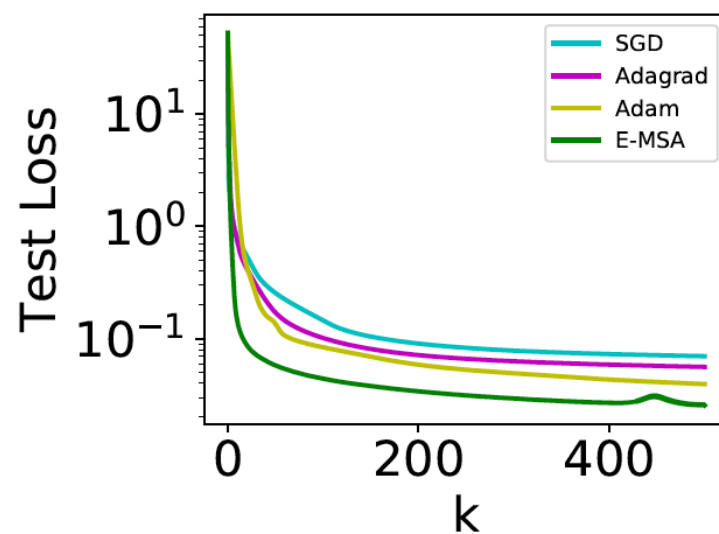
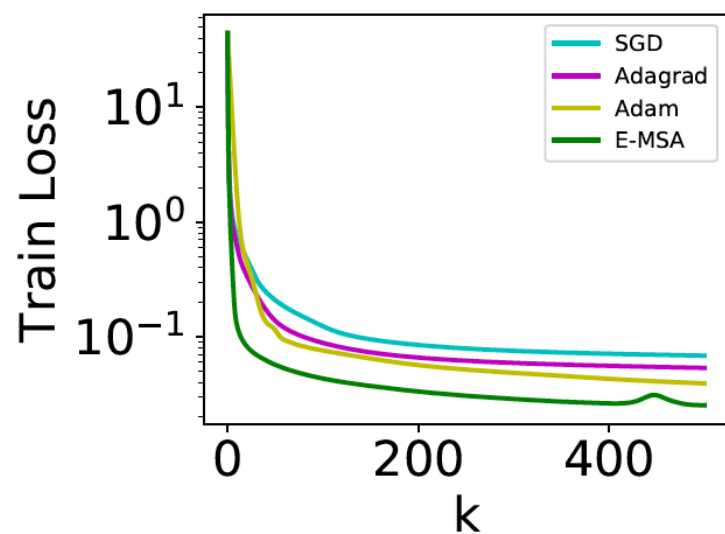
- Solve

$$\frac{d\mathbf{p}_\tau^k}{d\tau} = -\nabla_{\mathbf{z}} H(\mathbf{z}_\tau^k, \mathbf{p}_\tau^k, \theta_\tau^k), \quad \mathbf{p}_1^k = 2(f(\mathbf{x}; \theta^k) - f^*(\mathbf{x}))\mathbf{1}$$

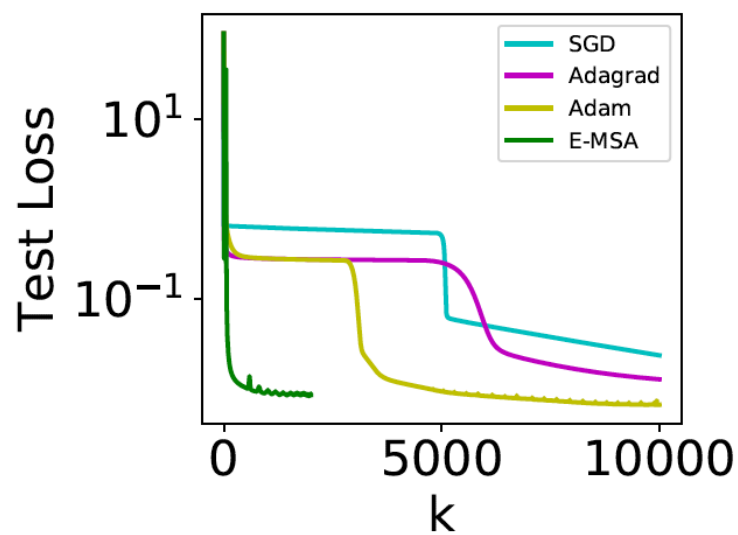
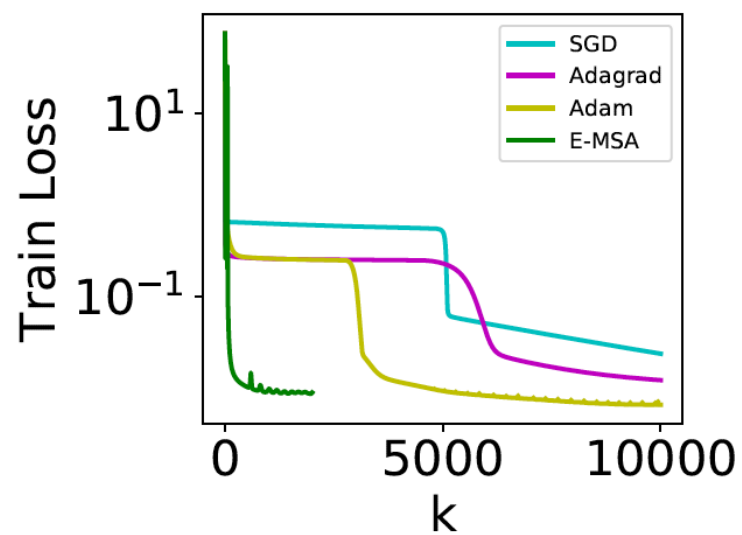
- Set $\theta_\tau^{k+1} = \operatorname{argmax}_{\theta \in \Theta} H(\mathbf{z}_\tau^k, \mathbf{p}_\tau^k, \theta)$, for each $\tau \in [0, 1]$

Extended MSA:

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



(a)



(b)

What have we really learned from ML?

Representation of functions as expectations:

- integral-transform based:

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

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

- flow-based:

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

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 applied mathematics, and mathematics.
- A reasonable mathematical picture for ML is emerging, from the perspective of **numerical analysis**.

At the heart of the mathematical theory for machine learning is **high dimensional 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
- Algorithms for solving high dimensional PDEs: From nonlinear Monte Carlo to machine learning
- Integrating machine learning with physics-based modeling