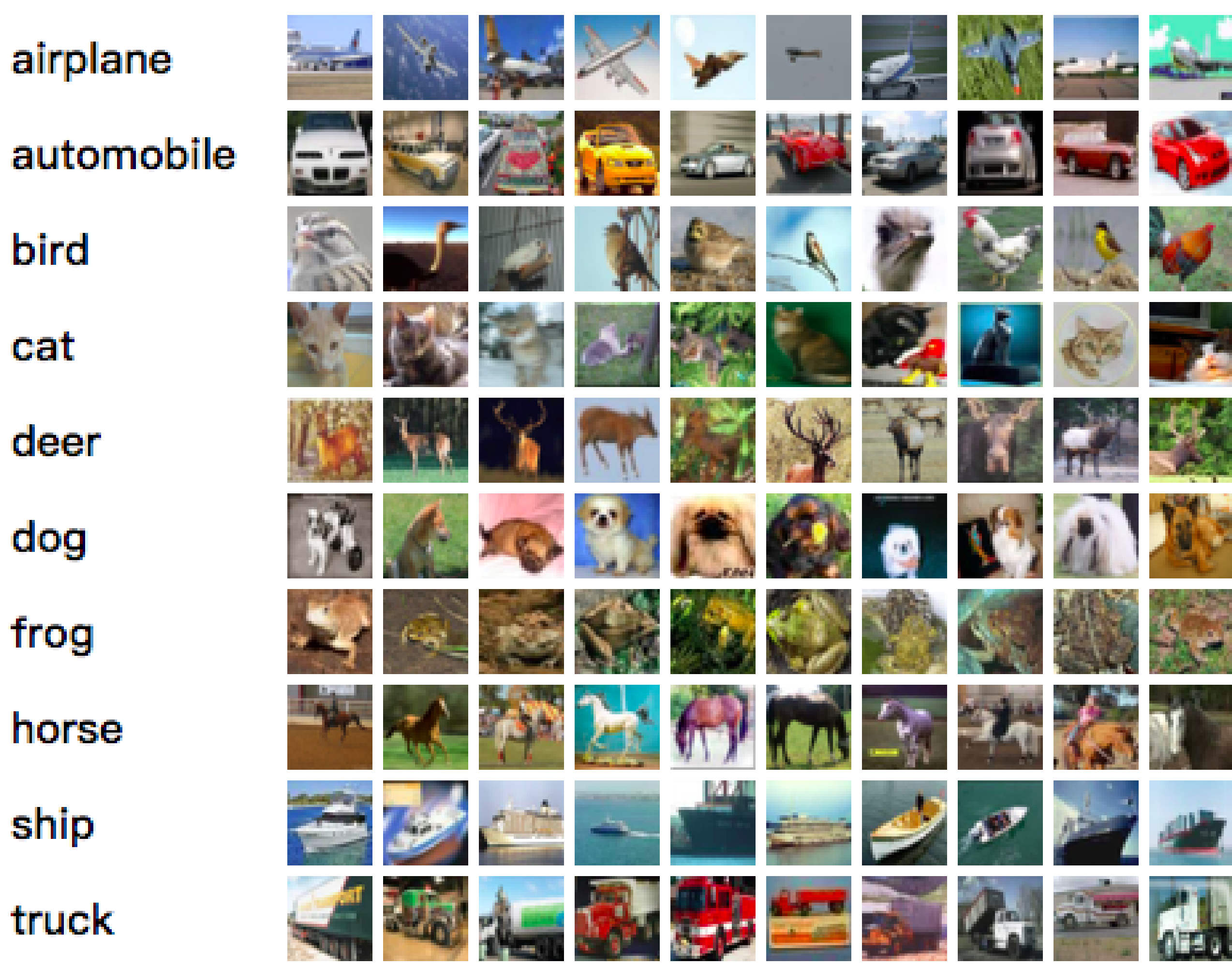
A Mathematical Perspective of Machine Learning

Weinan E

Peking University
Princeton University

ML can do wonders: Image classification

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$\}$.
- $f^* : [0,1]^{3072} \to \{$airplane, ... , 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.

AlphaGo

The math problem behind these examples:

- Image classification: Approximating functions
  Given $S = \{(x_j, y_j = f^*(x_j)), j \in [n]\}$, learn (i.e. approximate) f^*.
- Generating pictures of human face: Approximating probability distributions
- AlphaGo: approximating solutions of Bellman equations

Approximating functions, probability distributions and solving difference/differential equations are all standard problems in mathematics, particularly computational mathematics.

What is different here from what was done before is the high dimensionality.

For Cifar 10, each image is a point in $d = 32 \times 32 \times 3 = 3072$ dimensional Euclidean space.

Classical approximation theory: Approximating functions by polynomials or piecewise polynomials

Example: Piecewise linear approximation

- $d =$ dimensionality
- $m =$ number of free parameters
- $h =$ size of the mesh: $h \sim m^{-1/d}$

$$\inf_{f_m \in \mathcal{H}_m} \|f^* - f_m\|_{L^2(X)} \lesssim h \|\nabla^2 f^*\|_{L^2(X)}$$

- **Curse of Dimesionality (CoD)**: If we want to reduce the error by a factor of 10, we have to increase m by a factor of 10^d.

This is the case for all grid-based or fixed basis-based algorithms.

CoD is a common difficulty for a lot of problems!

- statistics and machine learning
- quantum many-body problem (solving Schrodinger's equation in quantum mechanics)
- classical many-body problem (e.g. protein folding)
- control theory/dynamic programming (Bellman 1958)
- combinatorial optimization (combinatorial explosion)

What did people do before deep learning?

- quantum mechanics: Hartree approximation

$$f(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_d) \sim f_1(\boldsymbol{x}_1) f_2(\boldsymbol{x}_2) \cdots f_d(\boldsymbol{x}_d)$$

  also called "approximate dynamic programming"
- generalized linear models

$$f(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_d) \sim \sum_{k=1}^{m} f_k(\boldsymbol{x}_1, \boldsymbol{x}_2, \cdots, \boldsymbol{x}_d)$$

  The set $\{f_k\}$ are called "features".

  Feature engineering: The construction of features using intuition, invariance, etc.

  Example: SIFT features

Deep learning seems to be a very powerful tool for attacking the CoD problem

- Can we use deep learning to solve other problems that suffer from CoD?
- Why is deep learning so powerful in high dimension?

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

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

   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):

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

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

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

Model dynamics (analog of ResNet):

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

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

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

Look for a feedback control:

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

- Neural network approximation:

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

Optimization problem (SGD applies directly)

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

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

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.

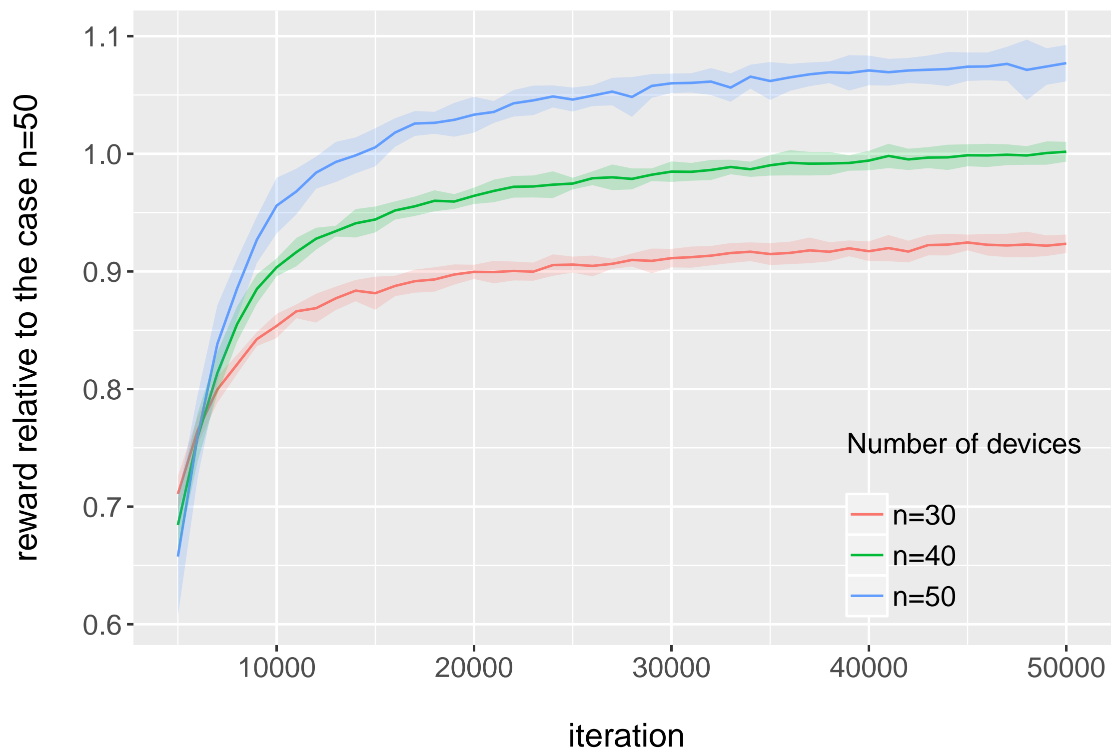


Figure: Relative reward. The space of control function is $\mathbb{R}^{n+2} \to \mathbb{R}^{3n}$ for $n = 30, 40, 50$, with multiple equality and inequality constrains.

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

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

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

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

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

The unique minimizer is the solution to the PDE with:

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

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:

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

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

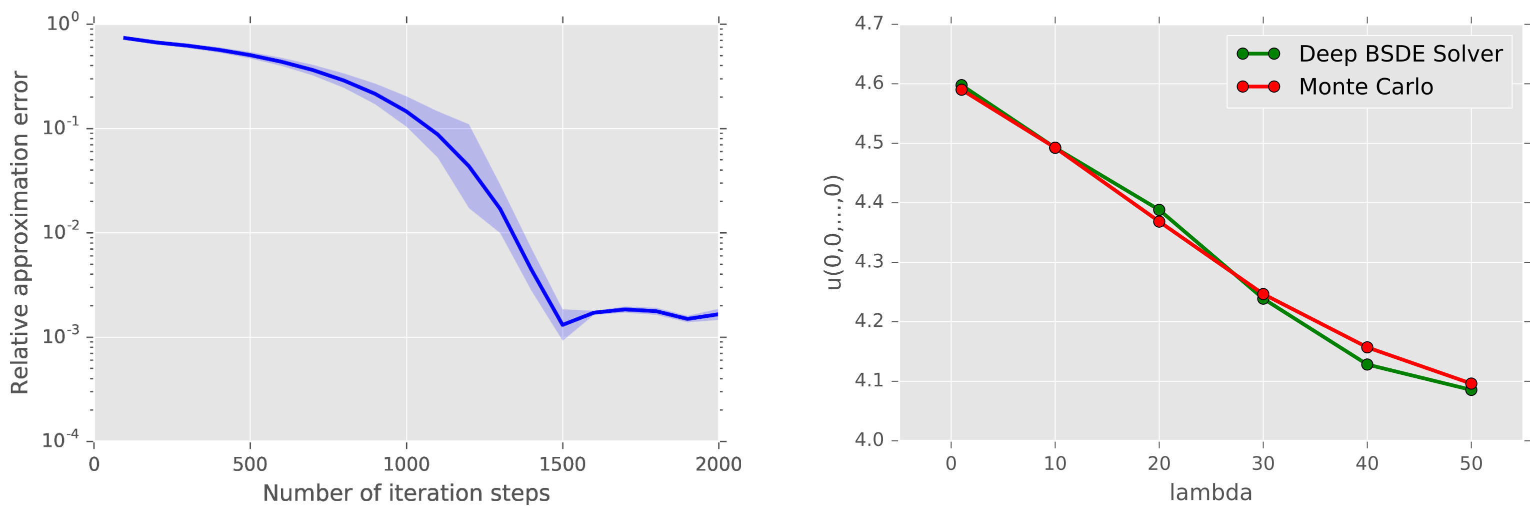
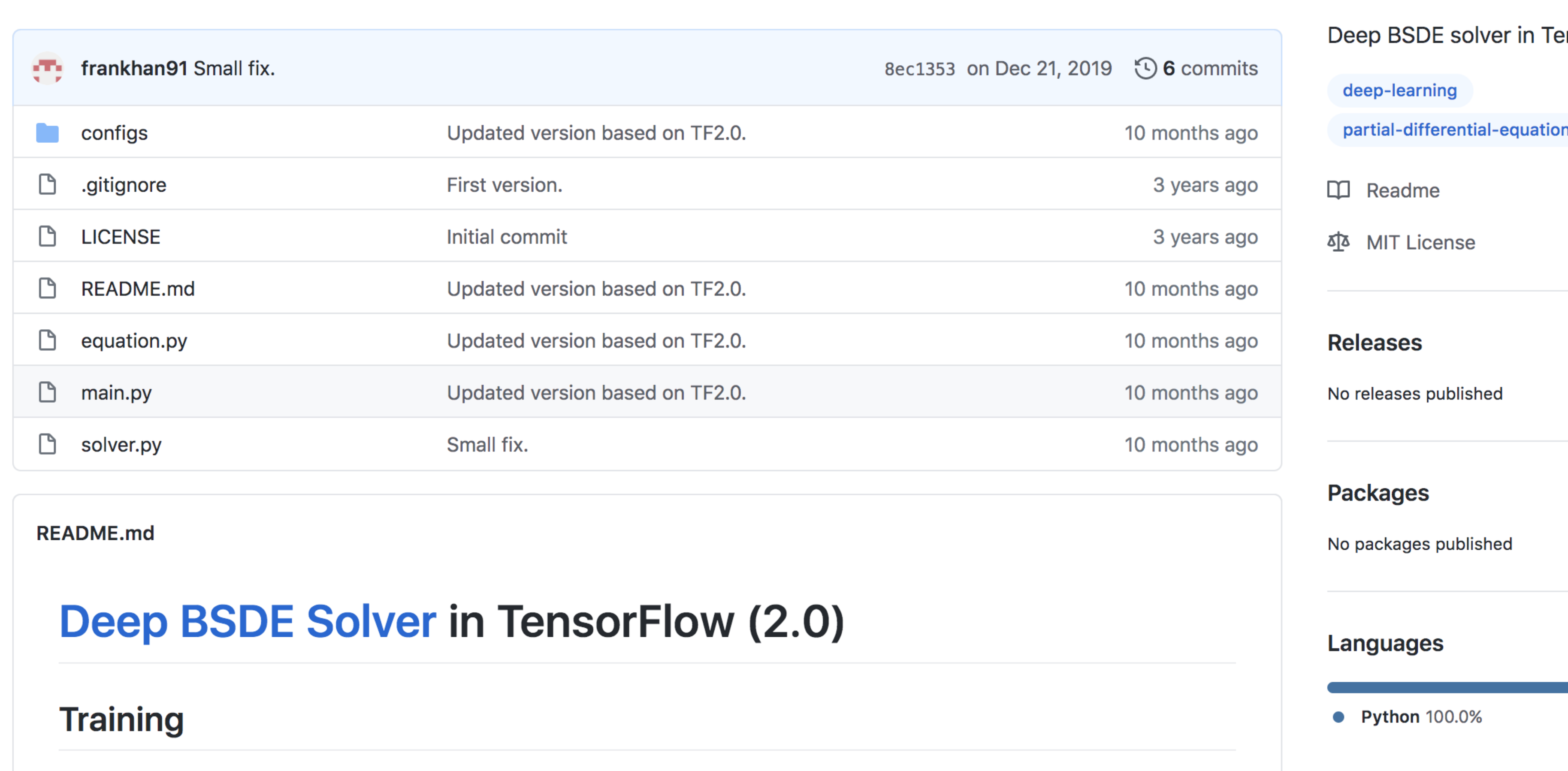


Figure: Left: Relative error of the deep BSDE method for $u(t{=}0, x{=}(0,\ldots,0))$ when $\lambda = 1$, which achieves 0.17% in a runtime of 330 seconds. Right: Optimal cost $u(t{=}0, x{=}(0,\ldots,0))$ against different λ.

Open source platform



Jiequn Han et al.

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

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

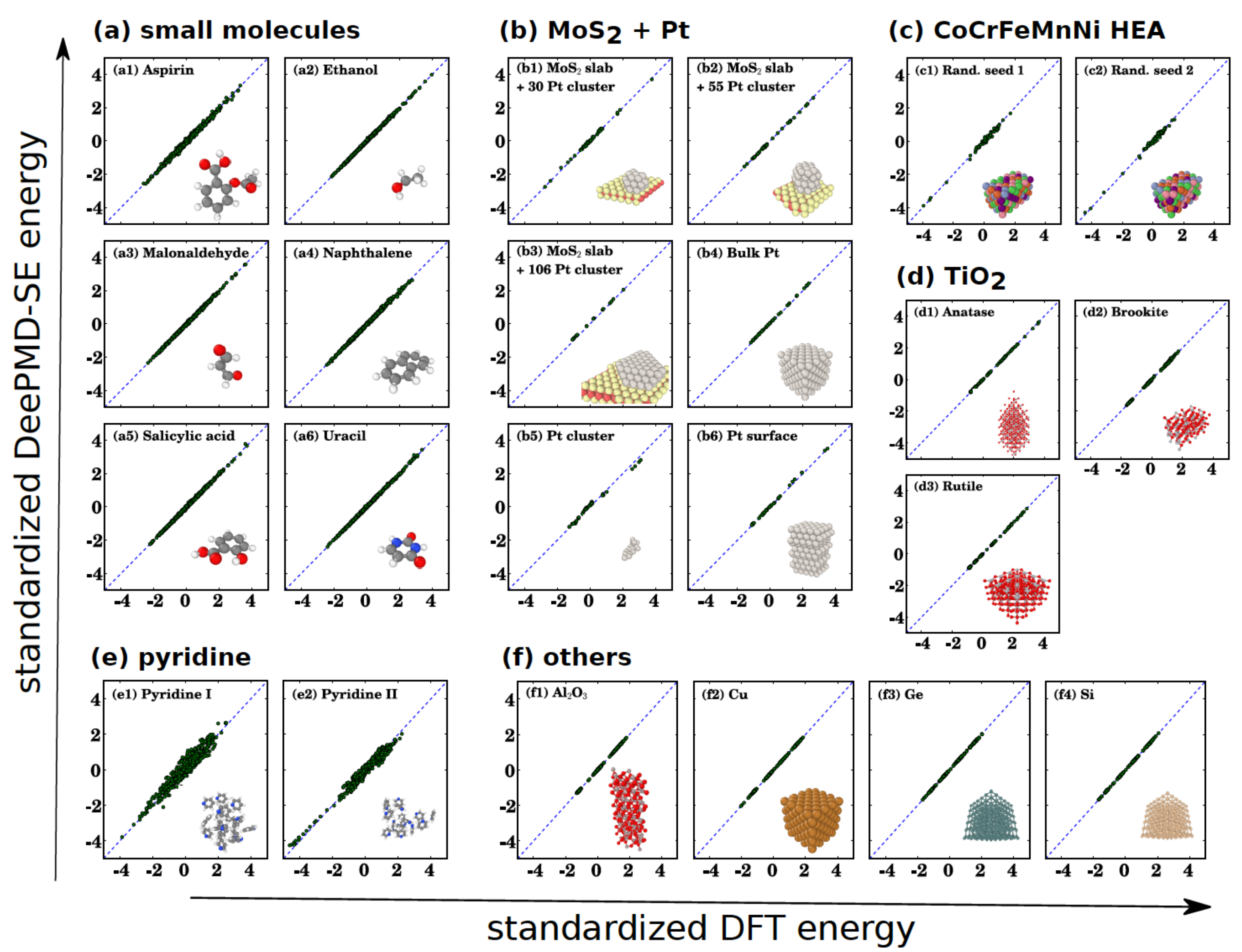
Key question: $V = ?$

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.

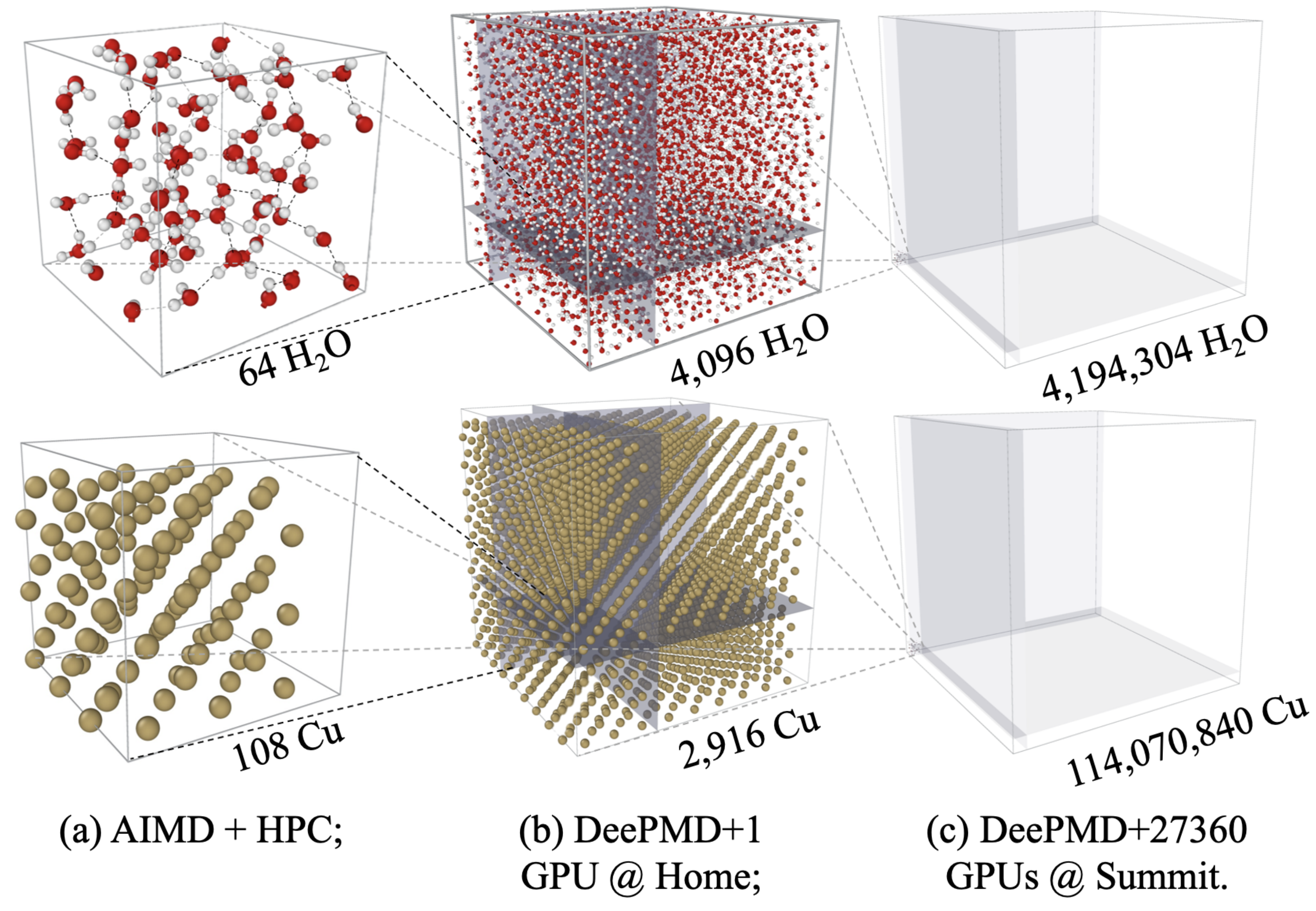
Now: Use QM to supply the data needed to train a neural network model.

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



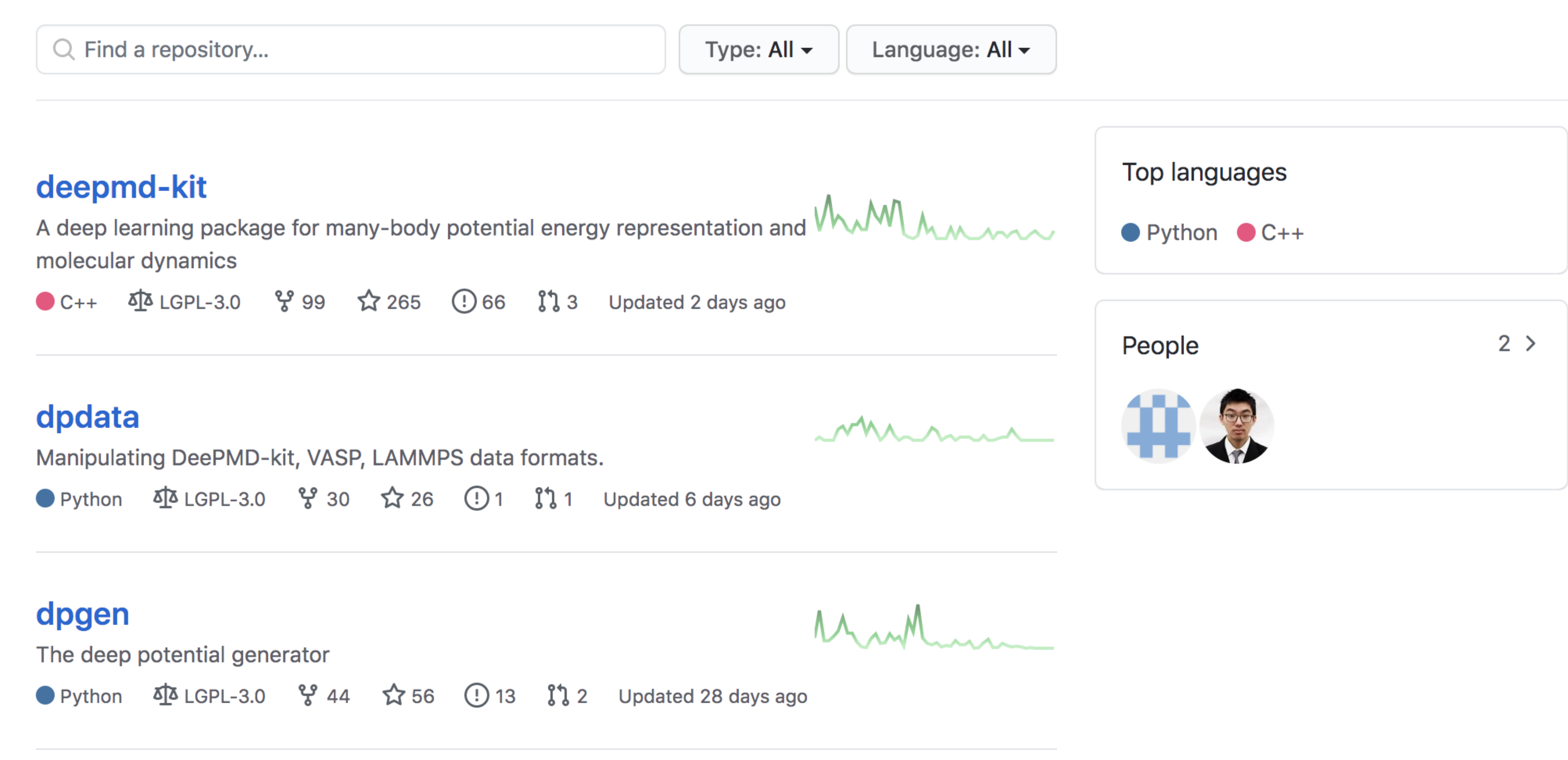
Linfeng Zhang, Jiequn Han, et al (2018)

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



Linfeng Zhang, Jiequn Han, Han Wang et al.

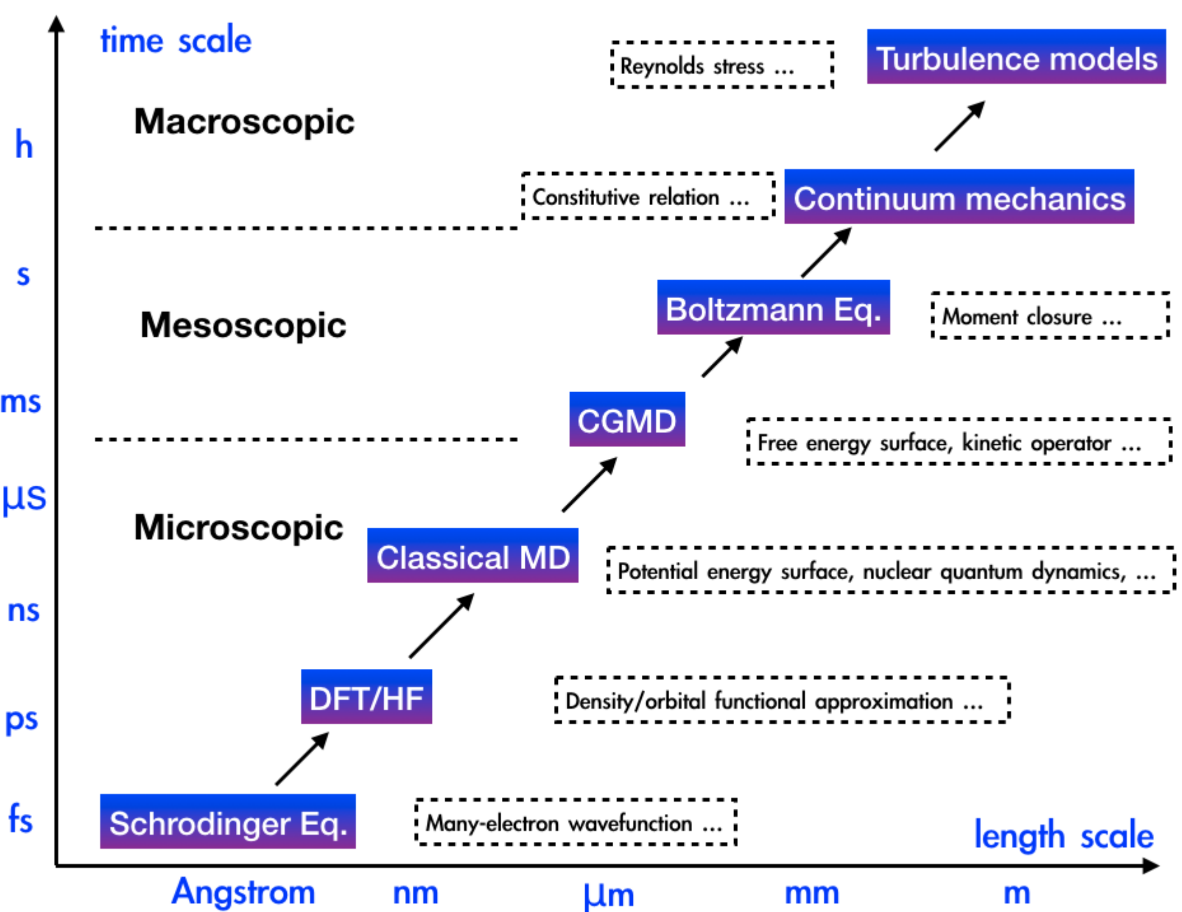


Figure: Representative physical models at different scale and their most important modeling ingredients.

Other work

- 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)), PINN
- 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: *Machine Learning-Assisted Modeling*, Physics Today, July, 2021.

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

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.

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

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

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

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

Approximating functions in high D: 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 $(\sigma(z) = e^{iz})$

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

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

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

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

Two-layer neural network model: Barron spaces

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

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

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

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

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

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

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

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 $\int_{\mathbb{R}^d} \|\boldsymbol{\omega}\|_1^2 |\hat{f}(\boldsymbol{\omega})| d\omega < \infty$, 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

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

- $\Gamma(f) = \int_{\mathbb{R}^d} \|\boldsymbol{\omega}\|_1^2 |\hat{f}(\boldsymbol{\omega})| d\omega$ is Barron's spectral norm. This condition depends on dimensionality.
- Let $f_1(\boldsymbol{x}) = \|\boldsymbol{x}\|$, $f_2(\boldsymbol{x}) = \text{dist}(\boldsymbol{x}, \mathcal{S}^{d-1})$, then f_1 is in $\mathcal{B}$, f_2 is not in $\mathcal{B}$.

Theorem (Direct Approximation Theorem)

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

Theorem (Inverse Approximation Theorem)

Let

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

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?

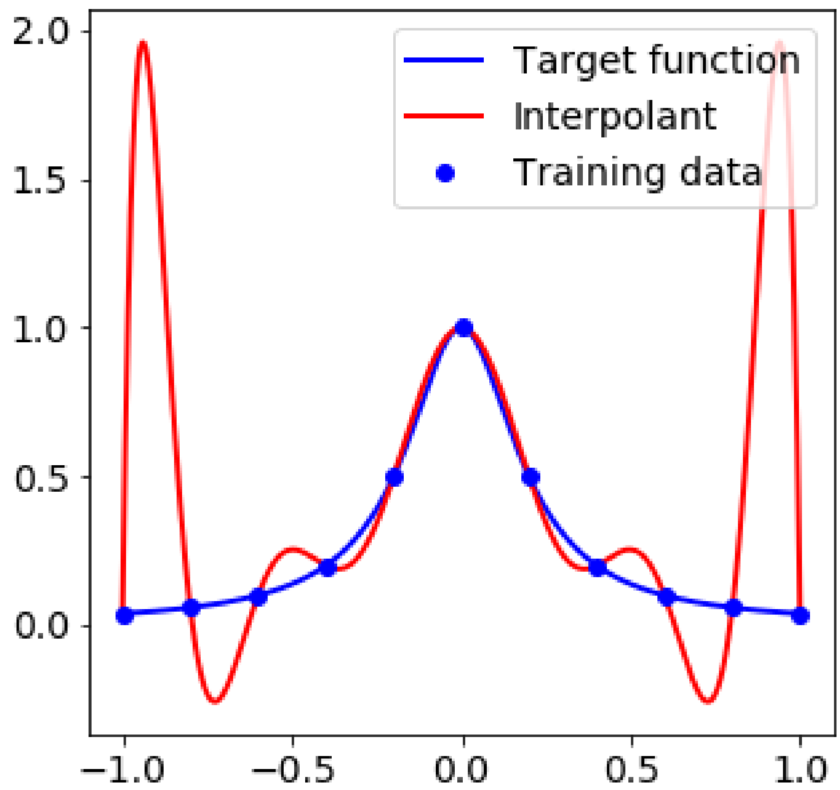


Figure: The Runge phenomenon: $f^*(x) = \frac{1}{1+25x^2}$

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.

In contrast, if $\mathcal{F} = \{f \in C^0, \|f\|_{\text{Lip}} \leq 1\}$, then

$$\mathrm{Rad}_S(\mathcal{F}) \leq \frac{C_d}{n^{1/d}}$$

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:

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

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

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

Optimization: Gradient flows

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

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

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

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

This is the same as the "mean-field" limit for 2-layer NNs.
It is also the Wasserstein-2 gradient flow for $\mathcal{R}$.

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:

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

Define

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

Claim: The PDE for ρ is equivalent to:

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

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

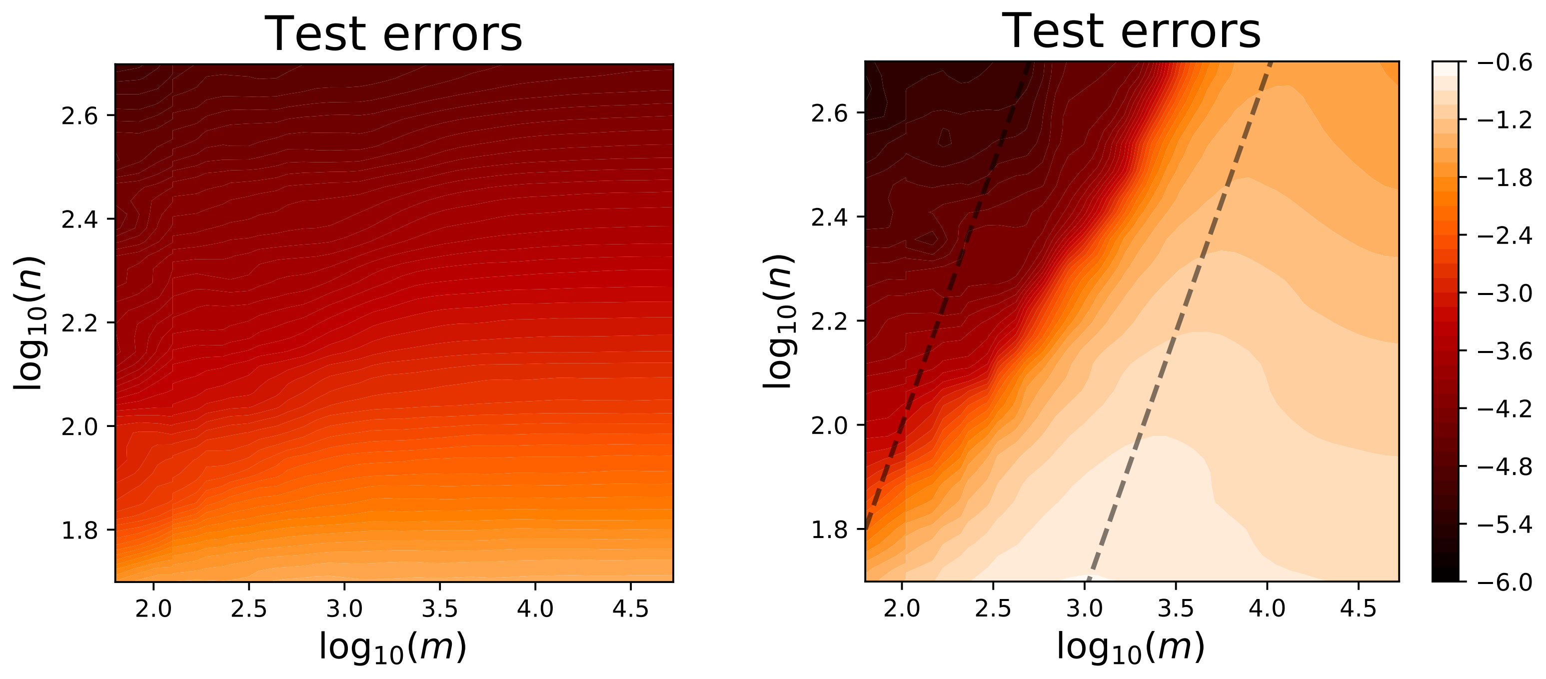


Figure: **(Left)** continuous viewpoint; **(Right)** conventional NN models. Target function is a single neuron.

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:

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

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

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

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

with the boundary conditions:

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

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

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

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

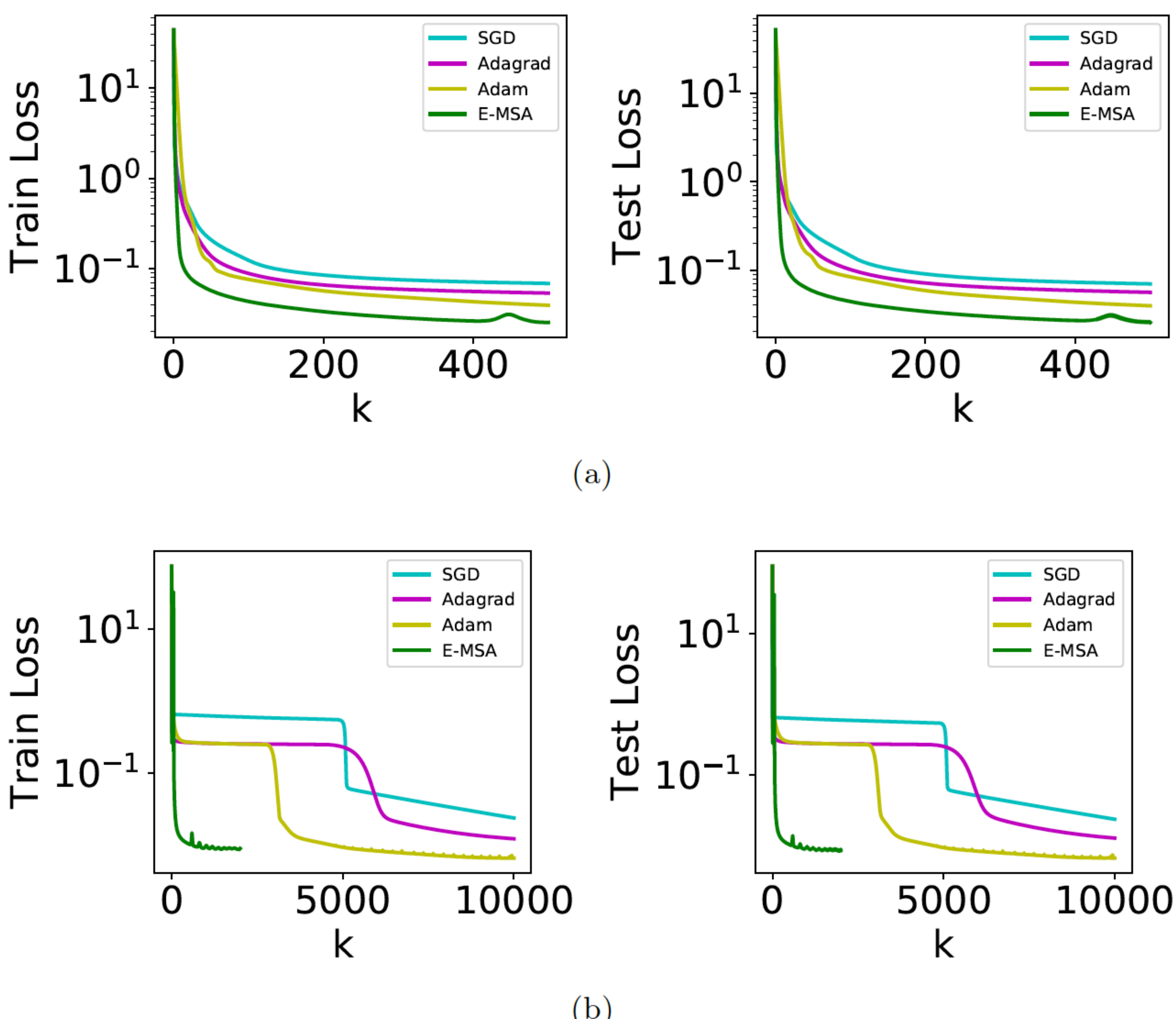
- Solve
$$\frac{d\boldsymbol{z}_\tau^k}{d\tau} = \nabla_{\boldsymbol{p}} H(\boldsymbol{z}_\tau^k, \boldsymbol{p}_\tau^k, \theta_\tau^k), \quad \boldsymbol{z}_0^k = V\boldsymbol{x}$$

- 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:

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



(a)

(b)

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 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