A Few Thoughts on Deep Learning-Based Scientific Computing

Haizhao Yang

Department of Mathematics
Purdue University

Inverse Problems Seminar
Department of Mathematics and Computer Science
University College London
February 5, 2021

Deep Learning for Scientific Computing?

Still not a complete story.

Outline

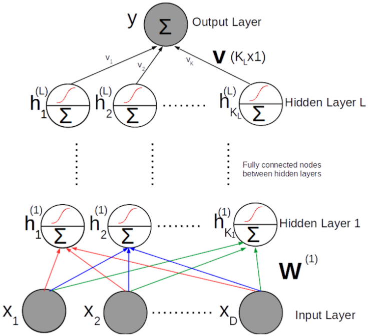
- Neural Network Approximation
  - Exponential Approximation Rate
  - Curse of dimensonality
  - Deep network is powerful
- Neural Network Optimization
  - Global convergence for supervised learning
  - Global convergence for solving PDEs
  - But assumption is strong
- Neural Network Generalization
  - Generalization for supervised learning
  - Generalization for solving PDEs
  - But requires regularization

Deep neural networks

$$y = h(x;\theta) := T \circ \phi(x) := T \circ h^{(L)} \circ h^{(L-1)} \circ \cdots \circ h^{(1)}(x)$$

where

- $h^{(i)}(x) = \sigma({W^{(i)}}^T x + b^{(i)})$;
- $T(x) = V^T x$;
- $\theta = (W^{(1)}, \cdots, W^{(L)}, b^{(1)}, \cdots, b^{(L)}, V)$.



Supervised deep learning

Conditions

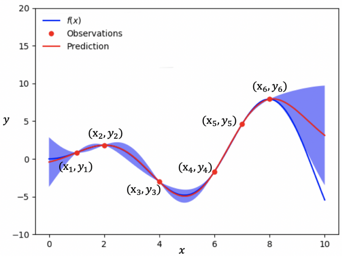
- Given data pairs $\{(x_i, y_i = f(x_i))\}$ from an unknown map $f(x)$ defined on Ω
- $\{x_i\}_{i=1}^n$ are sampled randomly from an unknown distribution $U(x)$ on Ω

Goal

Recover the unknown map $f(x)$

Deep learning

- Design a family of DNNs $\{h(x;\theta)\}_\theta$ of a given size
- Find the best DNN $h(x;\theta) \approx f(x)$ on Ω



Supervised deep learning

Deep learning ideally

- Quantify how good $h(x;\theta) \approx f(x)$ via the population loss:

$$R_D(\theta) \overset{\text{e.g.}}{=} \mathrm{E}_{x\sim U(\Omega)}\left[|h(x;\theta) - f(x)|^2\right]$$

- The best solution is $h(x;\theta_D)$ with

$$\theta_D = \operatorname{argmin} R_D(\theta)$$

- But $U(\Omega)$ is not known

Deep learning in practice

- Only the empirical loss is available:

$$R_S(\theta) := \frac{1}{N}\sum_{i=1}^{N}(h(x_i;\theta) - y_i)^2$$

- The best empirical solution is $h(x;\theta_S)$ with

$$\theta_S = \operatorname{argmin} R_S(\theta)$$

- Numerical optimization to obtain a numerical solution $h(x;\theta_N)$.
- In practice, $\theta_N \neq \theta_S \neq \theta_D$ and how good $R_D(\theta_N)$ is?

Supervised deep learning

A full error analysis of $R_D(\theta_N)$

$$\begin{aligned} R_D(\theta_N) = & [R_D(\theta_N) - R_S(\theta_N)] + [R_S(\theta_N) - R_S(\theta_S)] + [R_S(\theta_S) - R_S(\theta_D)] \\ & + [R_S(\theta_D) - R_D(\theta_D)] + R_D(\theta_D) \\ \leq & \, R_D(\theta_D) + [R_S(\theta_N) - R_S(\theta_S)] \\ & + [R_D(\theta_N) - R_S(\theta_N)] + [R_S(\theta_D) - R_D(\theta_D)], \end{aligned}$$

A full error analysis of $R_D(\theta_N)$

$$\begin{aligned} R_D(\theta_N) &= [R_D(\theta_N) - R_S(\theta_N)] + [R_S(\theta_N) - R_S(\theta_S)] + [R_S(\theta_S) - R_S(\theta_D)] \\ &\quad + [R_S(\theta_D) - R_D(\theta_D)] + R_D(\theta_D) \\ &\le R_D(\theta_D) + [R_S(\theta_N) - R_S(\theta_S)] \\ &\quad + [R_D(\theta_N) - R_S(\theta_N)] + [R_S(\theta_D) - R_D(\theta_D)], \end{aligned}$$

- $R_D(\theta_D) = \int_\Omega (h(x;\theta_D) - f(x))^2 d\mu(x) \le \int_\Omega (h(x;\tilde{\theta}) - f(x))^2 d\mu(x)$ can be bounded by a constructive approximation of $\tilde{\theta}$

A full error analysis of $R_D(\theta_N)$

$$\begin{aligned} R_D(\theta_N) = & [R_D(\theta_N) - R_S(\theta_N)] + [R_S(\theta_N) - R_S(\theta_S)] + [R_S(\theta_S) - R_S(\theta_D)] \\ & + [R_S(\theta_D) - R_D(\theta_D)] + R_D(\theta_D) \\ \leq & R_D(\theta_D) + [R_S(\theta_N) - R_S(\theta_S)] \\ & + [R_D(\theta_N) - R_S(\theta_N)] + [R_S(\theta_D) - R_D(\theta_D)], \end{aligned}$$

- $R_D(\theta_D) = \int_\Omega (h(x;\theta_D) - f(x))^2 d\mu(x) \leq \int_\Omega (h(x;\tilde{\theta}) - f(x))^2 d\mu(x)$ can be bounded by a constructive approximation of $\tilde{\theta}$
- $[R_S(\theta_N) - R_S(\theta_S)]$ is the optimization error

A full error analysis of $R_D(\theta_N)$

$$
\begin{aligned}
R_D(\theta_N) = & [R_D(\theta_N) - R_S(\theta_N)] + [R_S(\theta_N) - R_S(\theta_S)] + [R_S(\theta_S) - R_S(\theta_D)] \\
& + [R_S(\theta_D) - R_D(\theta_D)] + R_D(\theta_D) \\
\leq & R_D(\theta_D) + [R_S(\theta_N) - R_S(\theta_S)] \\
& + [R_D(\theta_N) - R_S(\theta_N)] + [R_S(\theta_D) - R_D(\theta_D)],
\end{aligned}
$$

- $R_D(\theta_D) = \int_\Omega (h(x;\theta_D) - f(x))^2 d\mu(x) \leq \int_\Omega (h(x;\tilde{\theta}) - f(x))^2 d\mu(x)$ can be bounded by a constructive approximation of $\tilde{\theta}$
- $[R_S(\theta_N) - R_S(\theta_S)]$ is the optimization error
- Other two terms are the generalization error

Deep Learning for Solving PDEs

Goals

Learning the solutions of high-dimensional and highly nonlinear PDEs

Challenges for traditional methods

- curse of dimensionality

Machine learning for PDEs

- Owens and Filkin, 1989; Lee and Kang, 1990; Dissanayake and Phan-Thien, 1994
- RBM, Quantum Many-Body Problem, Giuseppe Carleo, Matthias Troyer, 2016
- BSDE, Han et al, 2017
- DGM, Sirignano and Spiliopoulos, 2017
- Deep Ritz, E and Yu, 2017
- PINN, Raissi, Perdikaris, and Karniadakis, 2017

Least Square Methods

Neural networks + least square for PDEs (date back to 1990s),

$$\mathcal{D}(u) = f \quad \text{in } \Omega,$$
$$\mathcal{B}(u) = g \quad \text{on } \partial\Omega.$$

A DNN $\phi(\boldsymbol{x};\boldsymbol{\theta}^*)$ is constructed to approximate the solution $u(\boldsymbol{x})$ via

$$\begin{aligned}\boldsymbol{\theta}^* &= \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \\ &:= \operatorname*{argmin}_{\boldsymbol{\theta}} \|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|_2^2 + \lambda\|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\|_2^2\end{aligned}$$

Least Square Methods

We aim at the full error analysis:

- Approximation theory
- Optimization theory
- Generalization theory

Deep Network Approximation

Goals

- The curse of dimensionality exist? e.g., $\#$ parameters not $(\frac{1}{\epsilon})^d$
- Is exponential approximation rate available? e.g., $\#$ parameters $\log(\frac{1}{\epsilon})$

Why this goal?

- Computational efficiency especially in high dimension

Literature Review

Active research directions

Cybenko, 1989; Hornik et al., 1989; Barron, 1993; Liang and Srikant, 2016; Yarotsky, 2017; Poggio et al., 2017; Schmidt-Hieber, 2017; E and Wang, 2018; Petersen and Voigtlaender, 2018; Chui et al., 2018; Yarotsky, 2018; Nakada and Imaizumi, 2019; Gribonval et al., 2019; Gühring et al., 2019; Chen et al., 2019; Li et al., 2019; Suzuki, 2019; Bao et al., 2019; E et al., 2019; Opschoor et al., 2019; Yarotsky and Zhevnerchuk, 2019; Bölcskei et al., 2019; Montanelli and Du, 2019; Chen and Wu, 2019; Zhou, 2020; Montanelli et al., 2020, etc.

Literature Review

Functions spaces

- Continuous functions
- Smooth functions
- Functions with integral representations

ReLU DNNs, continuous functions $C([0,1]^d)$

ReLU; Fixed network width $O(N)$ and depth $O(L)$

- Nearly tight error rate $5\omega_f(8\sqrt{d}N^{-2/d}L^{-2/d})$ simultaneously in N and L with L^∞-norm. Shen, Y., and Zhang (CiCP, 2020)
- ω_f is the modulas of continuity
- Improved to a tight rate $O\left(\sqrt{d}\,\omega_f\left(\left(N^2L^2\log_3(N+2)\right)^{-1/d}\right)\right)$. Shen, Y., and Zhang (J Math Pures Appl, 2021)

Curse of dimensionality exists!

ReLU DNNs, smooth functions $C^s([0,1]^d)$

Does smoothness help?

ReLU; Fixed network width $O(N)$ and depth $O(L)$

- Nearly tight rate $85(s+1)^d 8^s \|f\|_{C^s([0,1]^d)} N^{-2s/d} L^{-2s/d}$ simultaneously in N and L with L^∞-norm
- Lu, Shen, Y., and Zhang (SIMA 2021)

The curse of dimensionality exists if s is fixed.

DNNs with advanced activation function

Sine-ReLU; Fixed width $O(d)$, varying depth L

- $\exp(-c_{r,d}\sqrt{L})$ with L^∞-norm for $C^r([0,1]^d)$
- Root exponential approximation rate achieved
- Curse of dimensionality is not clear
- arotsky and Zhevnerchuk, NeurIPS 2020

Floor and ReLU activation, width $O(N)$ and depth $O(dL)$, $C([0,1]^d)$

- Error rate $\omega_f(\sqrt{d}N^{-\sqrt{L}}) + 2\omega_f(\sqrt{d})N^{-\sqrt{L}}$ with L^∞-norm
- Merely based on the compositional structure of DNNs
- NO curse of dimensionality for many continuous functions
- Root exponential approximation rate
- Shen, Y., and Zhang (Neural Computation, 2020)

DNNs with advanced activation function

What if we use more activation functions?

Floor, Sign, and 2^x activation, width $O(N)$ and depth 3, $C([0,1]^d)$

- Error rate $\omega_f(\sqrt{d}2^{-N}) + 2\omega_f(\sqrt{d})2^{-N}$ with L^∞-norm
- Merely based on the compositional structure of DNNs
- NO curse of dimensionality for many continuous functions
- Exponential approximation rate
- Shen, Y., and Zhang (Neural Networks, 2021)

Further interpretation of our result

Explicit error bound

Floor, Sign, and 2^x activation, width $O(N)$ and depth 3, Hölder$([0,1]^d, \alpha, \lambda)$

- Error rate $3\lambda(2\sqrt{d})^{\alpha}2^{-\alpha N}$ with L^{∞}-norm
- NO curse of dimensionality
- Exponential approximation rate
- Shen, Y., and Zhang (Neural Networks, 2021)

Further interpretation of our result

Realistic consideration

- Constructive approximation requires f or exponentially many samples given
- Constructed parameters require high precision computation
- Floor and Sign are discontinuous functions leading to gradient vanishing

DNNs with advanced activation function

A continuous activation function without gradient vanishing

$$\sigma_1(x) = \left|x - 2\lfloor \tfrac{x+1}{2} \rfloor\right|,$$

$$\sigma_2(x) := \frac{x}{|x|+1},$$

$$\sigma(x) := \begin{cases} \sigma_1(x) & \text{for } x \in [0, \infty), \\ \sigma_2(x) & \text{for } x \in (-\infty, 0). \end{cases}$$

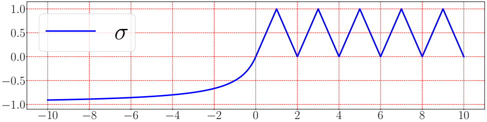


Figure: An illustration of σ on $[-10, 10]$.

Arbitrarily small error with a fixed number of neurons for $C([0,1]^d)$

- For any $\epsilon > 0$, there exists ϕ of width $36d(2d+1)$ and depth 11 s.t.
$$\|f(x) - \phi(x)\|_{L^\infty([0,1]^d)} \leq \epsilon$$
- Shen, Y., and Zhang (arXiv:2107.02397)

Exact representation with a fixed number of neurons for classification functions

- For any classification function $f(x)$ with K classes, there exists ϕ of width $36d(2d+1)$ and depth 12 s.t.

$$f(x) = \phi(x)$$

  on the supports of each class.
- Shen, Y., and Zhang (arXiv:2107.02397)

DNNs with advanced activation function

Two main ideas

- Kolmogorov-Arnold Superposition Theorem.

Theorem

$\forall f(\mathbf{x}) \in C([0,1]^d)$, there exist $\psi_p(x)$ and $\phi(x)$ in $C(\mathbb{R})$ and $b_{pq} \in \mathbb{R}$ s.t.

$$f(\mathbf{x}) = \sum_{q=1}^{2d+1} a_q \phi(\sum_{p=1}^{d} b_{pq}\psi_p(x_p)).$$

- NNs with width 36 and depth 5 is dense in $C([0,1])$ (Shen, Y., and Zhang (arXiv:2107.02397).

Summary

- Deep Neural Networks are powerful
- Quantitative approximation results are available
- How to quantify deep learning optimization and generalization errors?

Optimization and Generalization of Deep Learning

In the setting of supervised learning:

Mean-field analysis

- Chizat and Bach 2018; Mei et al. 2018; Mei et al. 2019, Lu et al. 2020, etc.
- Idea:
  1) a two-layer neural network can be seen as an approximation to an infinitely wide neural network with parameters following a distribution p_t;
  2) understanding network training via the evolution of p_t.

In the setting of solving PDEs: vastly open

Optimization and Generalization of Deep Learning

In the setting of supervised learning:

Neural tangent kernel/Lazy training

- Idea: in the limit of infinite width, deep learning becomes kernel methods
- Global optimization convergence:
  - Jacot et al. 2018 (two layers);
  - Du et al. 2019 (L layers, DNN);
  - Z Allen-Zhu, Y Li, Z Song 2018 (L layers, DNN, RNN);
  - D Zou*, Y Cao*, D. Zhou, and Q Gu 2018 (L layers, DNN, milder conditions)
  - Chizat et al. 2018
- Generalization theory
  - Y Cao and Q Gu, 2019a (GD)
  - Y Cao and Q Gu, 2019b (SGD)
- Consistent optimization and generalization for classification
  - Z Ji and M Telgarsky 2020
  - Z Chen*, Y Cao*, D Zou, and Q Gu 2020 (SOTA)

In the setting of solving PDEs: vastly open

Neural Tangent Kernel of Deep Learning Optimization

- **Optimization objective function**:

$$R_S(\boldsymbol{\theta}) := \frac{1}{N}\sum_{i=1}^{N}(h(\boldsymbol{x}_i;\boldsymbol{\theta}) - f(\boldsymbol{x}_i))^2$$

- Introduce $\mathcal{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N]^T \in \mathbb{R}^{N\times d}$, then
  - $h(\mathcal{X};\boldsymbol{\theta}(t)) := [h(\boldsymbol{x}_i;\boldsymbol{\theta}(t))] \in \mathbb{R}^N$
  - $\nabla_{\boldsymbol{\theta}} h(\mathcal{X};\boldsymbol{\theta}(t)) := [\nabla_{\boldsymbol{\theta}_j} h(\boldsymbol{x}_i;\boldsymbol{\theta}(t))] \in \mathbb{R}^{N\times W}$
  - $\nabla_{h(\mathcal{X};\boldsymbol{\theta}(t))} R_S := \frac{2}{N}(h(\mathcal{X};\boldsymbol{\theta}(t)) - f(\mathcal{X})) := [\frac{2}{N}(h(\boldsymbol{x}_i;\boldsymbol{\theta}(t)) - f(\boldsymbol{x}_i))] \in \mathbb{R}^N$
- **Gradient descent**

$$\begin{aligned}\boldsymbol{\theta}(t+1) &= \boldsymbol{\theta}(t) - \tau\frac{2}{N}\sum_{i=1}^{N}(h(\boldsymbol{x}_i;\boldsymbol{\theta}(t)) - f(\boldsymbol{x}_i))\nabla_{\boldsymbol{\theta}(t)} h(\boldsymbol{x}_i;\boldsymbol{\theta}) \\ &= \boldsymbol{\theta}(t) - \tau\nabla_{\boldsymbol{\theta}} h(\mathcal{X};\boldsymbol{\theta}(t))^T\nabla_{h(\mathcal{X};\boldsymbol{\theta}(t))} R_S,\end{aligned}$$

- **Gradient flow**

$$\partial_t\boldsymbol{\theta}(t) = -\nabla_{\boldsymbol{\theta}} h(\mathcal{X};\boldsymbol{\theta}(t))^T\nabla_{h(\mathcal{X};\boldsymbol{\theta}(t))} R_S,$$

Neural Tangent Kernel of Deep Learning Optimization

- **Gradient flow**

$$\partial_t \boldsymbol{\theta}(t) = -\nabla_{\boldsymbol{\theta}} h(\mathcal{X}; \boldsymbol{\theta}(t))^T \nabla_{h(\mathcal{X}; \boldsymbol{\theta}(t))} R_S,$$

- **DNN evolution**

$$\partial_t h(\mathcal{X}; \boldsymbol{\theta}(t)) = \nabla_{\boldsymbol{\theta}} h(\mathcal{X}; \boldsymbol{\theta}(t)) \partial_t \boldsymbol{\theta}(t) = -\hat{\Theta}_t(\mathcal{X}, \mathcal{X}) \nabla_{h(\mathcal{X}; \boldsymbol{\theta}(t))} R_S$$

  with the neural tangent kernel (NTK)

$$\hat{\Theta}_t = \nabla_{\boldsymbol{\theta}} h(\mathcal{X}; \boldsymbol{\theta}(t)) \nabla_{\boldsymbol{\theta}} h(\mathcal{X}; \boldsymbol{\theta}(t))^T.$$

- Nonlinear ODEs and challenging to analyze

Neural Tangent Kernel of Deep Learning Optimization

- **Linearization**

$h^{\text{lin}}(\boldsymbol{x};\theta(t)) := h(\boldsymbol{x};\theta(0)) + \nabla_{\boldsymbol{\theta}} h(\boldsymbol{x};\theta(0))(\theta(t)-\theta(0)) \approx h(\boldsymbol{x};\theta(t)),$

- **Approximate DNN evolution**

$$\begin{aligned} \partial_t h^{\text{lin}}(\boldsymbol{x};\boldsymbol{\theta}(t)) &= -\hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\nabla_{h^{\text{lin}}(\boldsymbol{x};\boldsymbol{\theta}(t))} R_S \\ &= -\hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\frac{2}{N}(h^{\text{lin}}(\boldsymbol{x};\boldsymbol{\theta}(t)) - f(\mathcal{X})) \end{aligned}$$

- **Linear ODE with a solution**

$h^{\text{lin}}(\boldsymbol{x};\theta(t)) = h(\boldsymbol{x};\theta(0)) - \hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\hat{\Theta}_0^{-1}\left(I - e^{-\hat{\Theta}_0 t}\right)(h(\mathcal{X};\boldsymbol{\theta}(0)) - \mathcal{Y})$

and

$$h^{\text{lin}}(\mathcal{X};\theta(t)) = \left(I - e^{-\hat{\Theta}_0 t}\right)\mathcal{Y} + e^{-\hat{\Theta}_0 t} h(\mathcal{X};\boldsymbol{\theta}(0)).$$

with $\mathcal{Y} := [y_1,\ldots,y_N]^T \in \mathbb{R}^N$.

Approximate DNN evolution

$$h^{\mathrm{lin}}(\boldsymbol{x};\boldsymbol{\theta}(t)) = h(\boldsymbol{x};\boldsymbol{\theta}(0)) - \hat{\Theta}_0(\boldsymbol{x},\mathcal{X})\hat{\Theta}_0^{-1}\left(I - e^{-\hat{\Theta}_0 t}\right)(h(\mathcal{X};\boldsymbol{\theta}(0)) - \mathcal{Y})$$

and

$$h^{\mathrm{lin}}(\mathcal{X};\boldsymbol{\theta}(t)) = \left(I - e^{-\hat{\Theta}_0 t}\right)\mathcal{Y} + e^{-\hat{\Theta}_0 t} h(\mathcal{X};\boldsymbol{\theta}(0))$$

Insight for numerical performance

- Spectral bias of deep learning (Rahaman et al, 2018; Xu et al, 2018, Cao et al, 2019)
- $\sin$ activation to lessen spectral bias (Tancik et al, 2020; Sitzmann et al, 2020)
- Wendland activation for non-singular NTK (Benson, Damle, and Townsend, 2020)
- Reproducing activation function to reduce the condition number of NTK (Liang, Lyu, Wang, **Y.**, 2021)

Optimization for PDE Solvers

Question: can we apply existing optimization analysis for PDE solvers?

A simple example

- Two-layer network: $\phi(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{k=1}^{N} a_k \sigma(\boldsymbol{w}_k^T \boldsymbol{x})$.
- A second order differential equation: $\mathcal{L}u = f$ with

$$\mathcal{L}u = \sum_{\alpha,\beta=1}^{d} A_{\alpha\beta}(\boldsymbol{x}) u_{x_\alpha x_\beta}.$$

- $f(\boldsymbol{x};\boldsymbol{\theta}) := \mathcal{L}\phi(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{k=1}^{N} a_k \boldsymbol{w}_k^T A(\boldsymbol{x}) \boldsymbol{w}_k \sigma''(\boldsymbol{w}_k^T \boldsymbol{x})$ to fit $f(\boldsymbol{x})$
- Much more difficult nonlinearity in $\boldsymbol{x}$ and $\boldsymbol{w}$ in the fitting than the original NN fitting.

Optimization for PDE Solvers

Assumption

- Two-layer network: $\phi(\boldsymbol{x};\boldsymbol{\theta}) = \sum_{k=1}^{N} a_k \sigma(\boldsymbol{w}_k^T \boldsymbol{x})$ on $[0,1]^d$.
- A second order differential equation: $\mathcal{L}u = f$ with

$$\mathcal{L}u = \sum_{\alpha,\beta=1}^{d} A_{\alpha\beta}(\boldsymbol{x}) u_{x_\alpha x_\beta} + \sum_{\alpha=1}^{d} b_\alpha(\boldsymbol{x}) u_{x_\alpha} + c(\boldsymbol{x}) u.$$

- $\mathcal{L}$ satisfies the condition: there exists $M \geq 1$ such that for all $\boldsymbol{x} \in \Omega = [0,1]^d$, $\alpha, \beta \in [d]$, we have $A_{\alpha\beta} = A_{\beta\alpha}$

$$|A_{\alpha\beta}(\boldsymbol{x})| \leq M, \quad |b_\alpha(\boldsymbol{x})| \leq M, \quad \text{and} \quad |c(\boldsymbol{x})| \leq M.$$

- Fixed n samples in the PDE domain.
- Empirical loss

$$R_S(\boldsymbol{\theta}) = \frac{1}{2n} \sum_{\{\boldsymbol{x}_i\}_{i=1}^{n}} |\mathcal{L}\phi(\boldsymbol{x}_i;\boldsymbol{\theta}) - f(\boldsymbol{x}_i)|^2$$

and population loss

$$R_{\mathcal{D}}(\boldsymbol{\theta}) = \frac{1}{2} \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}} \left[|\mathcal{L}\phi(\boldsymbol{x}_i;\boldsymbol{\theta}) - f(\boldsymbol{x}_i)|^2 \right]$$

with ϕ satisfying boundary conditions.

Luo and Y., preprint, 2020

Theorem (Linear convergence rate)

Let $\boldsymbol{\theta}^0 := \mathrm{vec}\{a_k^0, \boldsymbol{w}_k^0\}_{k=1}^N$ be the GD initialization, where $a_k^0 \sim \mathcal{N}(0, \gamma^2)$ and $\boldsymbol{w}_k^0 \sim \mathcal{N}(\boldsymbol{0}, \mathbb{I}_d)$ with any $\gamma \in (0, 1)$. Let $C_d := \mathbb{E}\|\boldsymbol{w}\|_1^{12} < +\infty$ with $\boldsymbol{w} \sim \mathcal{N}(\boldsymbol{0}, \mathbb{I}_d)$ and λ_S be a positive constant. For any $\delta \in (0, 1)$, if width

$$N \geq \max\left\{\frac{512 n^4 M^4 C_d}{\lambda_S^2 \delta}, \frac{200\sqrt{2} M d^3 n \log(4N(d+1)/\delta)\sqrt{R_S(\boldsymbol{\theta}^0)}}{\lambda_S}, \frac{2^{23} M^3 d^9 n^2 (\log(4N(d+1)/\delta))^4 \sqrt{R_S(\boldsymbol{\theta}^0)}}{\lambda_S^2}\right\},$$

then with probability at least $1-\delta$ over the random initialization $\boldsymbol{\theta}^0$, we have, for all $t \geq 0$,

$$R_S(\boldsymbol{\theta}(t)) \leq \exp\left(-\frac{N\lambda_S t}{n}\right) R_S(\boldsymbol{\theta}^0).$$

Generalization of PDE solvers

Luo and Y., preprint, 2020

Theorem (A posteriori generalization bound)

For any $\delta \in (0,1)$, with probability at least $1-\delta$ over the choice of random sample locations $S := \{\boldsymbol{x}_i\}_{i=1}^n$, for any two-layer neural network $\phi(\boldsymbol{x};\boldsymbol{\theta})$, we have

$$
\begin{aligned}
|R_{\mathcal{D}}(\boldsymbol{\theta}) - R_S(\boldsymbol{\theta})| \quad \leq \quad & \frac{(\|\boldsymbol{\theta}\|_{\mathcal{P}}+1)^2}{\sqrt{n}} 2M^2 \Big(14d^2\sqrt{2\log(2d)} \\
& + \log[\pi(\|\boldsymbol{\theta}\|_{\mathcal{P}}+1)] + \sqrt{2\log(1/3\delta)}\Big)
\end{aligned}
$$

Proof: $|R_{\mathcal{D}}(\boldsymbol{\theta}) - R_S(\boldsymbol{\theta})| \leq$ Rademacher complexity + Stat error
$\leq O\left(\frac{\|\boldsymbol{\theta}\|_{\mathcal{P}}}{\sqrt{n}}\right) + O\left(\frac{1}{\sqrt{n}}\right)$

Generalization of PDE solvers

Regression: E, Ma, and Wu, CMS, 2019
PDE solvers: Luo and Y., preprint, 2020

Theorem (A priori generalization bound)

Suppose that $f(\boldsymbol{x})$ is in the Barron-type space $\mathcal{B}([0,1]^d)$ and $\lambda \geq 4M^2[2 + 14d^2\sqrt{2\log(2d)} + \sqrt{2\log(2/3\delta)}]$. Let

$$\boldsymbol{\theta}_{S,\lambda} = \arg\min_{\boldsymbol{\theta}} J_{S,\lambda}(\boldsymbol{\theta}) := R_S(\boldsymbol{\theta}) + \frac{\lambda}{\sqrt{n}}\|\boldsymbol{\theta}\|_{\mathcal{P}}^2 \log[\pi(\|\boldsymbol{\theta}\|_{\mathcal{P}} + 1)].$$

Then for any $\delta \in (0,1)$, with probability at least $1-\delta$ over the choice of random samples $S := \{\boldsymbol{x}_i\}_{i=1}^n$, we have

$$\begin{aligned} R_{\mathcal{D}}(\boldsymbol{\theta}_{S,\lambda}) &:= \mathbb{E}_{\boldsymbol{x}\sim\mathcal{D}}\frac{1}{2}(\mathcal{L}\phi(\boldsymbol{x};\boldsymbol{\theta}_{S,\lambda}) - f(\boldsymbol{x}))^2 \\ &\leq \frac{6M^2\|f\|_{\mathcal{B}}^2}{N} + \frac{\|f\|_{\mathcal{B}}^2 + 1}{\sqrt{n}}(4\lambda + 16M^2)\left\{\log[\pi(2\|f\|_{\mathcal{B}} + 1)] \right. \\ &\left. + 14d^2\sqrt{\log(2d)} + \sqrt{\log(2/3\delta)}\right\}. \end{aligned}$$

Proof: $R_{\mathcal{D}}(\boldsymbol{\theta}_{S,\lambda}) \leq$ Approximation error + Rademacher complexity + Stat error $\leq O\left(\frac{\|f\|_{\mathcal{B}}^2}{N}\right) + O\left(\frac{\|\boldsymbol{\theta}\|_{\mathcal{P}}}{\sqrt{n}}\right) + O\left(\frac{1}{\sqrt{n}}\right) \leq O\left(\frac{\|f\|_{\mathcal{B}}^2}{N}\right) + O\left(\frac{\|f\|_{\mathcal{B}}^2}{\sqrt{n}}\right)$

Acknowledgment

Collaborators

Qiang Du, Yiqi Gu, Jianguo Huang, Senwei Liang, Jianfeng Lu, Tao Luo, Liyao Lyu, Hadrien Montanelli, Zuowei Shen, Chunmei Wang, Haoqin Wang, Chunmei Wang, Shijun Zhang, Chao Zhou

Funding

National Science Foundation under the grant award 1945029

Key ideas of our approximation

For $\boldsymbol{x} \in Q_{\boldsymbol{\beta}}$:
$$\boldsymbol{x} \to \phi_1(\boldsymbol{x}) = \boldsymbol{\beta} \to \phi_2(\boldsymbol{\beta}) = k_{\boldsymbol{\beta}} \to \phi_3(k_{\boldsymbol{\beta}}) = f(\boldsymbol{x}_{\boldsymbol{\beta}}) \approx f(\boldsymbol{x})$$

- Piecewise constant approximation: $f(\boldsymbol{x}) \approx f_p(\boldsymbol{x}) \approx \phi_3 \circ \phi_2 \circ \phi_1(\boldsymbol{x})$
- 2^N pieces per dim and 2^{Nd} pieces with accuracy 2^{-N}
- Floor NN $\phi_1(\boldsymbol{x})$ s.t. $\phi_1(\boldsymbol{x}) = \boldsymbol{\beta}$ for $\boldsymbol{x} \in Q_{\boldsymbol{\beta}}$ and $\boldsymbol{\beta} \in \mathbb{Z}^d$.
- Linear NN ϕ_2 mapping $\boldsymbol{\beta}$ to an integer $k_{\boldsymbol{\beta}} \in \{1, \ldots, 2^{Nd}\}$
- Key difficulty: NN ϕ_3 of width $O(N)$ and depth $O(1)$ fitting 2^{Nd} samples in 1D with accuracy $O(2^{-N})$
- ReLU NN fails

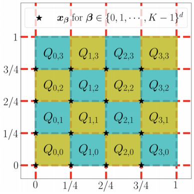


Figure: Uniform domain partitioning.

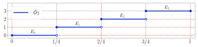


Figure: Floor function.

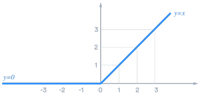


Figure: ReLU function.

Key ideas of our approximation

Binary representation and approximation

$\theta = \sum_{\ell=1}^{\infty} \theta_\ell 2^{-\ell}$ with $\theta_\ell \in \{0, 1\}$ is approximated by $\sum_{\ell=1}^{N} \theta_\ell 2^{-\ell}$ with an error 2^{-N}.

Bit extraction via a floor NN of width 2 and depth 1

$$\phi_k(\theta) := \lfloor 2^k \theta \rfloor - 2\lfloor 2^{k-1}\theta \rfloor = \theta_k$$

Bit extraction via a floor NN of width $2N$ and depth 1

Given $\theta = \sum_{\ell=1}^{\infty} \theta_\ell 2^{-\ell}$

$$\phi(\theta) := \begin{pmatrix} \phi_1(\theta) \\ \vdots \\ \phi_N(\theta) \end{pmatrix} = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_N \end{pmatrix} \in \mathbb{Z}^N$$

Key ideas of our approximation

Encoding K numbers to one number

- Extract bits $\{\theta_1^{(k)}, \ldots, \theta_N^{(k)}\}$ from $\theta^{(k)} = \sum_{\ell=1}^{\infty} \theta_\ell^{(k)} 2^{-\ell}$ for $k = 1, \ldots, K$
- sum up to get $a = \sum_{\ell=1}^{N} \theta_\ell^{(1)} 2^{-\ell} + \sum_{\ell=N+1}^{2N} \theta_\ell^{(2)} 2^{-\ell} + \cdots + \sum_{\ell=(K-1)N+1}^{KN} \theta_\ell^{(K)} 2^{-\ell}$

Decoding one number to get the k-th numbers

- Extract bits $\{\theta_1^{(k)}, \ldots, \theta_N^{(k)}\}$ from a via

$$\psi(k) := \phi(2^{(k-1)N} a - \lfloor 2^{(k-1)N} a \rfloor)$$

  of width $O(N)$ and depth $O(1)$.
- sum up to get $\theta^{(k)} \approx \sum_{\ell=1}^{N} \theta_\ell^{(k)} 2^{-\ell} = [2^{-1}, \ldots, 2^{-N}] \psi(k) := \gamma(k)$,
- $\gamma(k)$ is an NN of width $O(N)$ and depth $O(1)$.

Key Lemma

There exists an NN γ of width $O(N)$ and depth $O(1)$ that can memorize arbitrary samples $\{(k, \theta^{(k)}\}_{k=1}^{K}$ with a precision 2^{-N}.

Key ideas of our approximation

For $\boldsymbol{x} \in Q_{\boldsymbol{\beta}}$:

$$\boldsymbol{x} \to \phi_1(\boldsymbol{x}) = \boldsymbol{\beta} \to \phi_2(\boldsymbol{\beta}) = k_{\boldsymbol{\beta}} \to \phi_3(k_{\boldsymbol{\beta}}) = f(\boldsymbol{x}_{\boldsymbol{\beta}}) \approx f(\boldsymbol{x})$$

- Piecewise constant approximation: $f(\boldsymbol{x}) \approx f_p(\boldsymbol{x}) \approx \phi_3 \circ \phi_2 \circ \phi_1(\boldsymbol{x})$
- 2^N pieces per dim and 2^{Nd} pieces with accuracy 2^{-N}
- Floor NN $\phi_1(\boldsymbol{x})$ s.t. $\phi_1(\boldsymbol{x}) = \boldsymbol{\beta}$ for $\boldsymbol{x} \in Q_{\boldsymbol{\beta}}$ and $\boldsymbol{\beta} \in \mathbb{Z}^d$.
- Linear NN ϕ_2 mapping $\boldsymbol{\beta}$ to an integer $k_{\boldsymbol{\beta}} \in \{1, \ldots, 2^{Nd}\}$
- Key difficulty: NN ϕ_3 of width $O(N)$ and depth $O(1)$ fitting 2^{Nd} samples in 1D with accuracy $O(2^{-N})$
- Key Lemma: There exists an NN γ of width $O(N)$ and depth $O(1)$ that can memorize arbitrary samples $\{(k, \theta^{(k)})\}_{k=1}^{K}$ with a precision 2^{-N}.

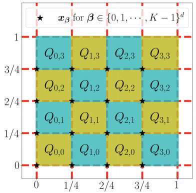


Figure: Uniform domain partitioning.

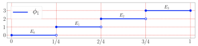


Figure: Floor function.

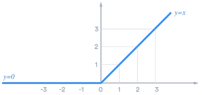


Figure: ReLU function.