A Few Thoughts on Deep Learning-Based Scientific Computing

Haizhao Yang
Department of Mathematics
Purdue University

Computational and Applied Mathematics Colloquium
Department of Statistics
University of Chicago
November 20, 2020

Acknowledgment

Collaborators

Qiang Du, Yiqi Gu, Jianguo Huang, Jianfeng Lu, Hadrien Montanelli, Zuowei Shen, Chunmei Wang, Haoqin Wang, Shijun Zhang, Chao Zhou

Funding

National Science Foundation under the grant award 1945029

Deep Learning for Scientific Computing?

- Good to use?
- How to use?

Outline

- Deep Network Approximation
  - Exponential Convergence
  - Curse of dimensonality
  - Deep network is powerful
- Deep Network Optimization
  - Global convergence for supervised learning
  - Global convergence for solving PDEs
  - Deep learning optimization can be realistic?
- Practical applications in scientific computing
  - Approximate solutions for high-dimensional problems
  - Preconditioners for low-dimensional nonlinear problems
  - Multiple solutions of nonlinear problems

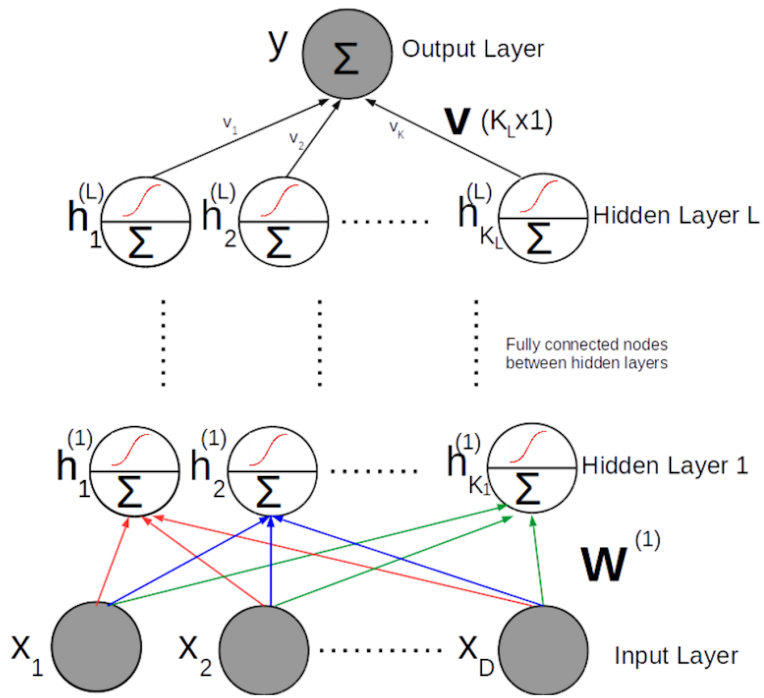
Deep Neural Network

Function composition in the parametrization:

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

where

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



Deep Learning for Solving PDEs

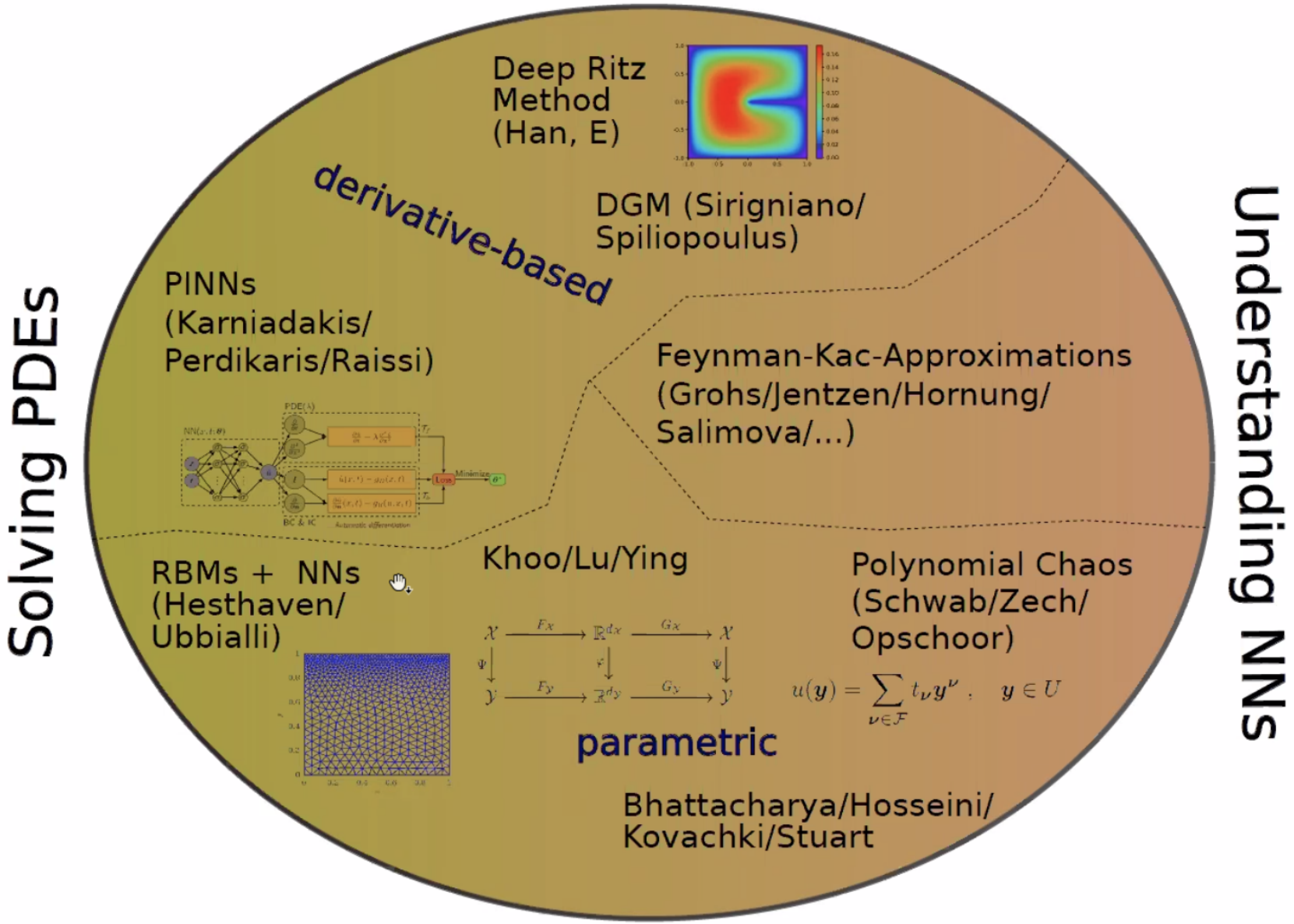


Figure: Figure by Phillip Peterson.

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

$$\begin{aligned}
\boldsymbol{\theta}^* &= \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \\
&:= \operatorname*{argmin}_{\boldsymbol{\theta}} \|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})\|_2^2 + \lambda \|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})\|_2^2 \\
&= \operatorname*{argmin}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\in\Omega}\left[|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})|^2\right] + \lambda \mathbb{E}_{\boldsymbol{x}\in\partial\Omega}\left[|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})|^2\right].
\end{aligned}$$

Stochastic gradient descent method

- Randomly generate sample sets Ω^r and $\partial\Omega^r$
- Define a random loss function

$$\begin{aligned}\mathcal{L}(\boldsymbol{\theta}, \Omega^r, \partial\Omega^r) \quad := \quad & \frac{1}{|\Omega^r|} \sum_{\boldsymbol{x}\in\Omega^r} \left[|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})|^2 \right] \\ & + \frac{\lambda}{|\partial\Omega^r|} \sum_{\boldsymbol{x}\in\partial\Omega^r} \left[|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})|^2 \right].\end{aligned}$$

- Update via gradient descent

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \frac{\partial \mathcal{L}(\boldsymbol{\theta}, \Omega^r, \partial\Omega^r)}{\partial \boldsymbol{\theta}}$$

Main foundation of deep learning

- Approximation theory: how good DNNs approximating functions?
- Optimization algorithms: how can we obtain (nearly) the best parameters?
- Generalization analysis: generalize well to unseen data?

Numerical observation

- Non-uniqueness of powerful network representation.
- Many good local minimizers.
- Fast convergence to approximate solutions.

Gaps between theory and practice.

Deep Network Approximation

Goals

- The curse of dimensionality exist?
- Is exponential convergence available?

Why this goal?

- Computational efficiency
- For example, when we apply DNNs to solve high-dimensional PDEs, it is better to answer the above questions.

Literature Review

A long list with 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.
- Functions spaces: continuous functions, smooth functions, functions with integral representations;
- Tools: polynomial approximations, the law of large number, Kolmogorov-Arnold representation theory, bit extraction technology (Bartlett et al., 1998; Harvey et al., 2017).

Our understanding on the literature

ReLU DNNs

- Curse of dimensionality exisits for continuous and smooth functions.
- Exponential convergence is achievable for special function classes.

DNNs with advanced activation functions

- Curse of dimensionality does not exisit and exponential convergence is achievable for (Hölder) continuous functions.
- Require exponentially large computation with high precision.

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

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

- Tight error rate $O(L^{-2/d})$ with L^∞-norm
- Yarotsky, 2018

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

- Tight error rate $5\omega_f(8\sqrt{d}N^{-2/d}L^{-2/d})$ simultaneously in N and L with L^∞-norm
- ω_f is the modulas of continuity
- Shen, Y., and Zhang (CiCP, 2020)

Curse of dimensionality exists!

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

Does smoothness help?

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

- Tight error rate $O(L^{-2s/d})$ with L^∞-norm
- Yarotsky, 2019

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

- 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 (preprint, 2020)

The curse of dimensionality exists if s is fixed and smoothness may help a bit

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 convergence achieved
- Curse of dimensionality is not clear
- Yarotsky, 2019

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

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)

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 (preprint, 2020)

Open problem: How to train these networks efficiently?

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

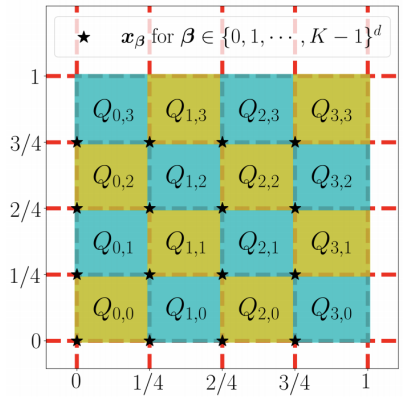


Figure: Uniform domain partitioning.

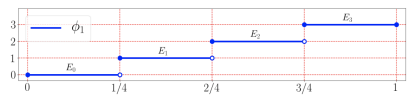


Figure: Floor function.

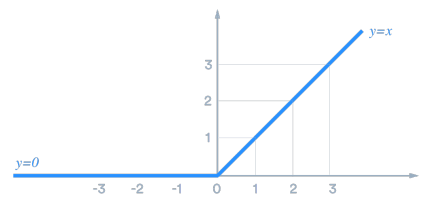


Figure: ReLU function.

Summary

- Deep Neural Networks are powerful
- How to identify these networks via optimization?

Why Deep Neural Netwokrs?

From the point of view of optimization.

Conjectures

- When network is sufficiently large, local minimizers are usually good minimizers.

Reality

- No practical theory and even efficient algorithm.

Neural tangent kernel/Lazy training

- Jacot et al. 2018; Du et al. 2019a; Allen-Zhu et al. 2019b; Du et al. 2019b; Zou et al. 2018; Chizat et al. 2019, etc.
- Idea: in the limit of infinite width, DNN becomes kernel methods

Mean-field analysis

- Chizat and Bach 2018; Mei et al. 2018; Mei et al. 2019, Lu et al. 2020
- 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.

Key Analysis of Neural Tangent Kernel

Simplifying the residual dynamic via approximation:

$$\begin{aligned}
\phi(\boldsymbol{X};\boldsymbol{\theta}_{t+1}) - f(\boldsymbol{X}) &\approx [\boldsymbol{I} - \tfrac{m\eta}{n}\boldsymbol{H}_t](\phi(\boldsymbol{X};\boldsymbol{\theta}_t) - f(\boldsymbol{X})) && \text{(NN dynamic)} \\
&\approx [\boldsymbol{I} - \tfrac{m\eta}{n}\boldsymbol{H}_0](\phi(\boldsymbol{X};\boldsymbol{\theta}_t) - f(\boldsymbol{X})) && \text{(lazy training)} \\
&\approx [\boldsymbol{I} - \tfrac{m\eta}{n}\boldsymbol{H}](\phi(\boldsymbol{X};\boldsymbol{\theta}_t) - f(\boldsymbol{X})) && \text{(NTK dynamic)}
\end{aligned}$$

- Training samples $\boldsymbol{X} = [\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n]^T$;
- Learning rate η;
- Width m;
- Gram matrix $\boldsymbol{H}_t := (\frac{1}{m}\langle \nabla\phi(\boldsymbol{x}_i;\boldsymbol{\theta}_t), \nabla\phi(\boldsymbol{x}_j;\boldsymbol{\theta}_t)\rangle)_{n\times n}$;
- NTK $\boldsymbol{H} = \lim_{m\to\infty} \boldsymbol{H}_0$;
- $\approx$ valid when $\boldsymbol{\theta}_{t+1} \approx \boldsymbol{\theta}_t \leftarrow \eta \approx 0$;
- $\approx$ valid when $\boldsymbol{\theta}_t \approx \boldsymbol{\theta}_0 \leftarrow \eta \approx 0$;
- $\approx$ valid when $m \to \infty$ by the law of large numbers.

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

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

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

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

Theorem: optimization convergence (T. Luo and Y., preprint, 2020)

Under the assumption of over-parametrization, with a high probability over random initialization, gradient descent dynamics makes the empirical loss function linearly converge to zero.

Summary

- Deep network approximation is powerful
- Deep network optimization for scientific computing is challenging with few theories
  - Finite width and finite depth: open problem
  - SGD instead of gradient flow analysis: open problem
  - Nonlinear PDEs: open problem

Next question: Where to apply deep learning in scientific computing?

Deep learning for solving PDEs

Problem

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

Continuous method

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

$$\begin{aligned} \boldsymbol{\theta}^* &= \operatorname*{argmin}_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}) \\ &= \operatorname*{argmin}_{\boldsymbol{\theta}} \mathbb{E}_{\boldsymbol{x}\in\Omega}\left[|\mathcal{D}\phi(\boldsymbol{x};\boldsymbol{\theta}) - f(\boldsymbol{x})|^2\right] + \lambda \mathbb{E}_{\boldsymbol{x}\in\partial\Omega}\left[|\mathcal{B}\phi(\boldsymbol{x};\boldsymbol{\theta}) - g(\boldsymbol{x})|^2\right]. \end{aligned}$$

Our philosophy

- Non-uniqueness of network representation.
- Many good local minimizers.
- Fast convergence to good approximate solutions.

Example 1: Solving High-Dimensional PDEs

Previous successes

Han et al., Sirignano et al., Berg et al., Khoo et al., Maissi et al., etc.

SelectNet: Self-Paced Learning for High-dimensional Partial Differential Equations

- Gu, Y., and Zhou (preprint, 2019)
- Importance sampling
- Faster convergence
- Better local minimizers

SelectNet

Given a PDE problem,

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

- First DNN $\phi(\boldsymbol{x};\boldsymbol{\theta})$ is constructed to approximate the solution $u(\boldsymbol{x})$;
- Second DNN $\phi_s(\boldsymbol{x};\boldsymbol{\theta}_s) \in [m_1, m_2]$ for weighting.

$$\begin{aligned}\min_{\boldsymbol{\theta}} \max_{\boldsymbol{\theta}_s} \mathcal{L}(\boldsymbol{\theta},\boldsymbol{\theta}_s) : \quad = \quad & \mathrm{E}_{\boldsymbol{x}\in\Omega}\left[\phi_s(\boldsymbol{x};\boldsymbol{\theta}_s)\,|\mathcal{D}(\phi(\boldsymbol{x};\boldsymbol{\theta})) - f(\boldsymbol{x})|^2\right]\\ & +\lambda \mathrm{E}_{\boldsymbol{x}\in\partial\Omega}\left[\phi_s(\boldsymbol{x};\boldsymbol{\theta}_s)\,|\mathcal{B}(\phi(\boldsymbol{x};\boldsymbol{\theta})) - g(\boldsymbol{x})|^2\right]\\ & -\varepsilon^{-1}\left(\frac{1}{|\Omega|}\int_{\Omega}\phi_s(\boldsymbol{x};\boldsymbol{\theta}_s) - 1\right)^2\\ & -\varepsilon^{-1}\left(\frac{1}{|\partial\Omega|}\int_{\partial\Omega}\phi_s(\boldsymbol{x};\boldsymbol{\theta}_s) - 1\right)^2\end{aligned}$$

Nonlinear Elliptic Equations

- PDE

$$-\nabla \cdot (a(x)\nabla u) + |\nabla u|^2 = f(x), \quad \text{in } \Omega := \{\boldsymbol{x} : |\boldsymbol{x}| < 1\},$$
$$u = g(x), \quad \text{on } \partial\Omega,$$

  with $a(x) = 1 + \frac{1}{2}|\boldsymbol{x}|^2$.

- Example:

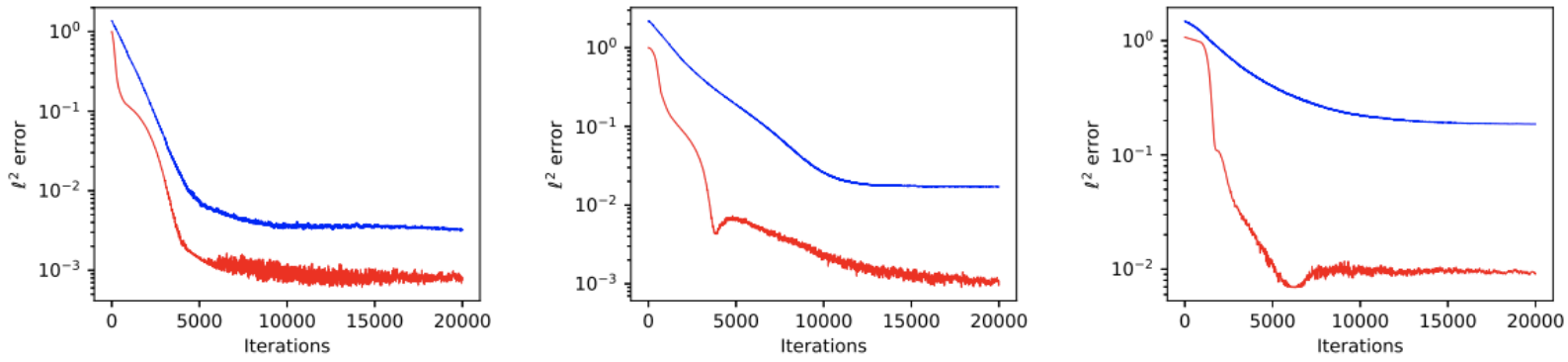
$$u(\boldsymbol{x}) = \sin(\frac{\pi}{2}(1 - |\boldsymbol{x}|)^{2.5})$$

  in $\Omega = \{\boldsymbol{x} : |\boldsymbol{x}| < 1\}$.

dimension	SelectNet	Basic
$d = 10$	7.9×10^{-4}	3.2×10^{-3}
$d = 20$	9.6×10^{-4}	1.7×10^{-2}
$d = 100$	9.3×10^{-3}	1.9×10^{-1}

Table: The minimal relative L^2 errors obtained by various methods in the nonlinear elliptic example.

Nonlinear Elliptic Equations



(a) $d = 10$ (b) $d = 20$ (c) $d = 100$

Fig. 6.2: *ℓ^2 errors v.s. iterations in the nonlinear elliptic example (Red: SelecNet model; Blue: the basic model).*

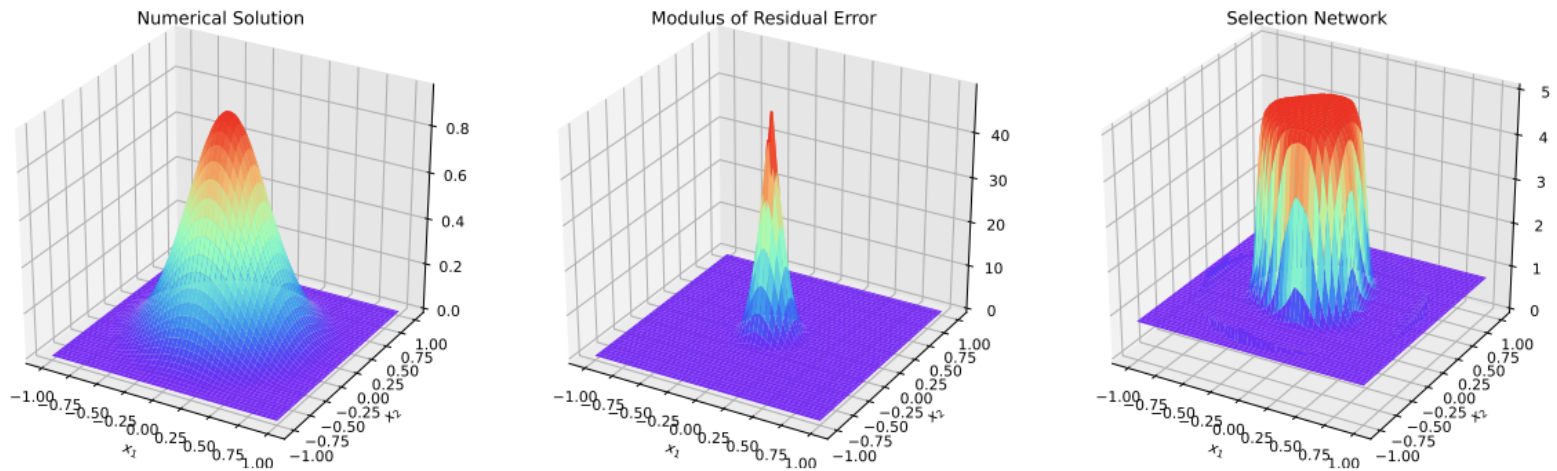


Fig. 6.3: *The (x_1, x_2)-surfaces of the numerical solution, modulus of residual error and selection network by SelectNet (d=20) in the nonlinear elliptic example.*

Parabolic Equations

- PDE

$$\begin{aligned} \partial_t u(x,t) - \Delta u(x,t) &= f(x,t), \quad \text{in } Q := \Omega \times (0,1), \\ u(x,t) &= g(x), \quad \text{on } \partial\Omega \times (0,1), \\ u(x,0) &= h(x), \quad \text{in } \Omega, \end{aligned}$$

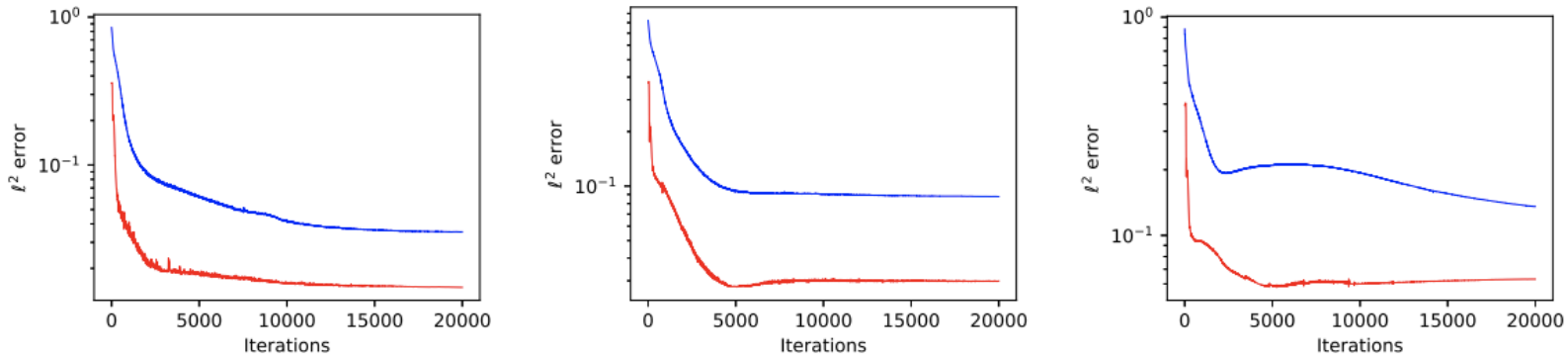
  with $\Omega := [-1,1]^d$.

- Example: $u(x,t) = e^{-t}\sin(\frac{\pi}{2}(1-|x|)^{2.5})$.

dimension	SelectNet	Basic
$d = 10$	1.5×10^{-2}	3.5×10^{-2}
$d = 20$	3.0×10^{-2}	8.7×10^{-2}
$d = 100$	6.3×10^{-2}	1.4×10^{-1}

Table: The minimal relative L^2 errors obtained by various methods in the parabolic example.

Parabolic Equations



(a) $d = 10$ (b) $d = 20$ (c) $d = 100$

Fig. 6.4: *ℓ^2 errors v.s. iterations in the first parabolic example (Red: SelecNet model; Blue: the basic model).*

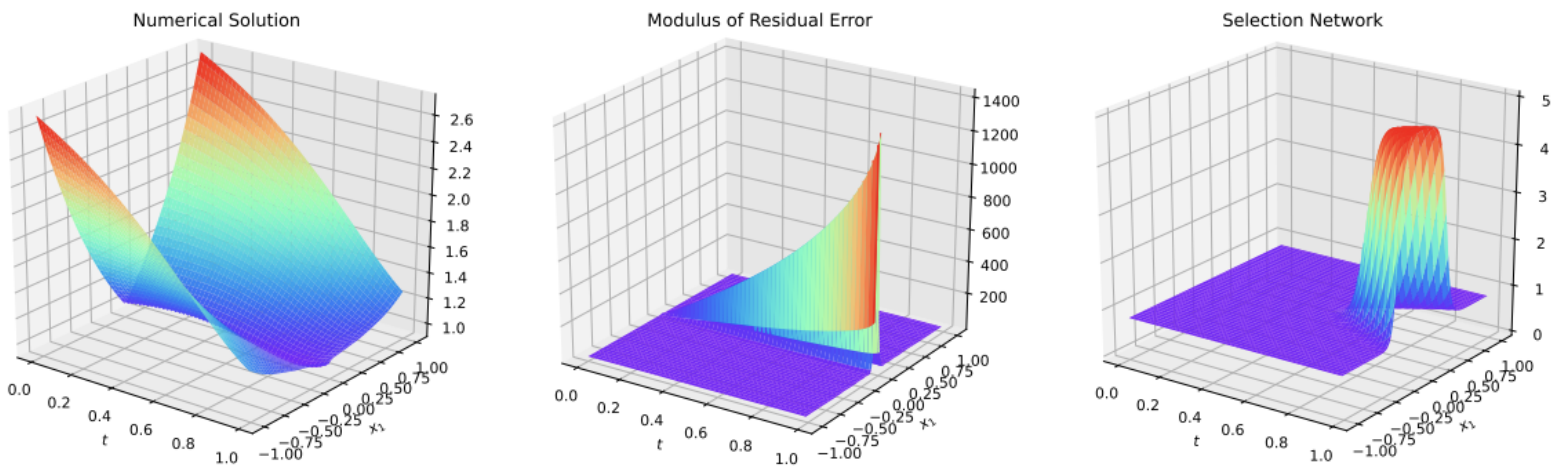
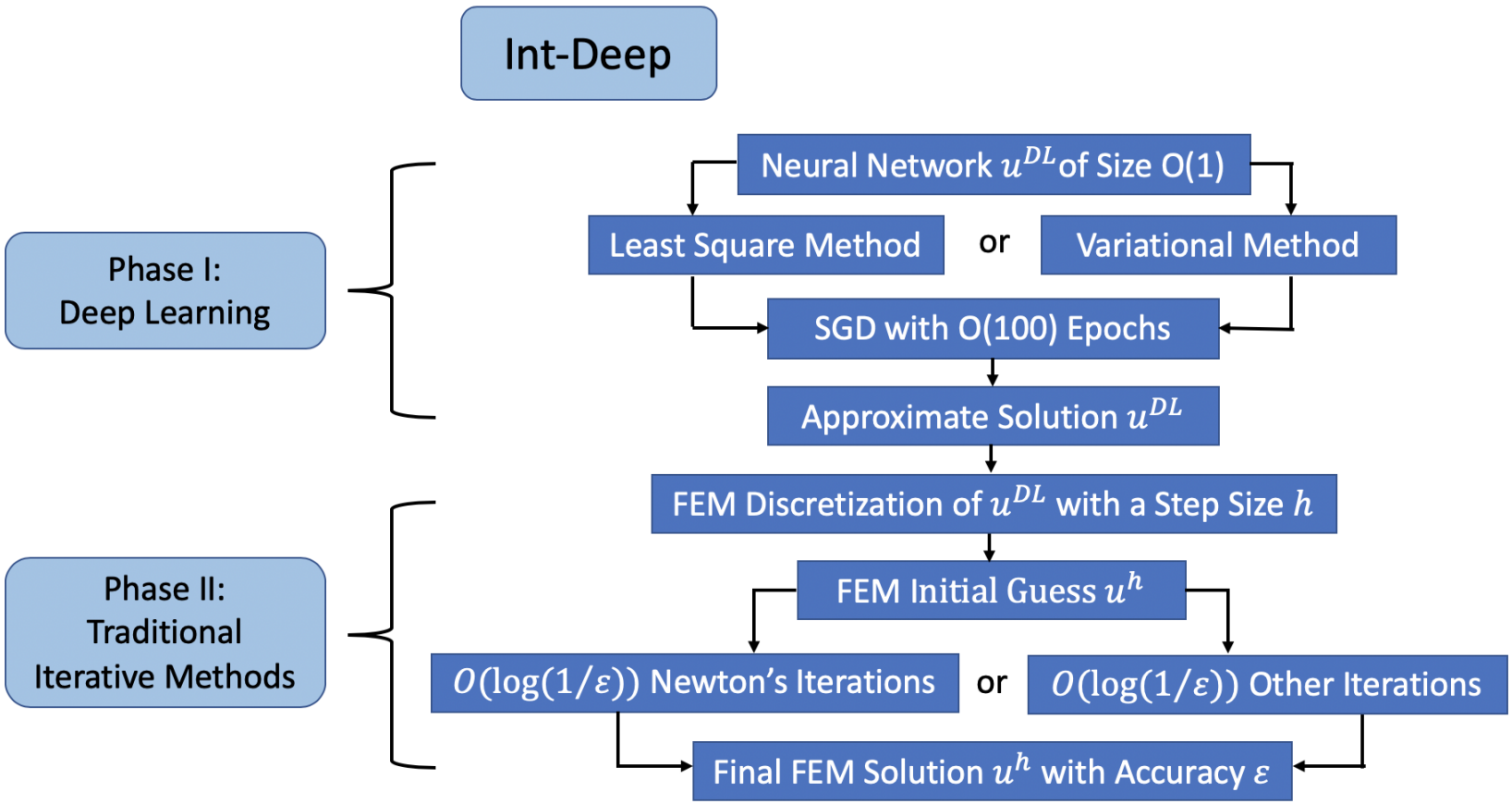


Fig. 6.5: *The (t, x_1)-surfaces of the numerical solution, modulus of residual error and selection network by SelectNet (d=20) in the first parabolic example.*

Example 2: Preconditioning

Int-Deep: A Deep Learning Initialized Iterative Method for Nonlinear Problems. Huang, Wang, and Y. (JCP, 2020)

- Idea: DL for initial guesses in traditional iterative methods



Semilinear elliptic equations

- PDE

$$\begin{cases} -\Delta u - (u-1)^3 + (u+2)^2 = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

  where $\Omega = (0,1)^d$.
- Example: $u(x) = 3\sin(2\pi x)$ for $d = 1$

Table: The performance of Int-Deep in 1D with different mesh sizes h with linear polynomials in FEM.

h	e_{h^2}	# Epoch	e^{DL}	#K	e^h	order
2^{-7}	6.1e-5	200	3.0e-1	5	1.2e-3	-
2^{-8}	1.5e-5	200	3.0e-1	5	2.9e-4	2.0
2^{-9}	3.8e-6	200	3.0e-1	5	7.3e-5	2.0
2^{-10}	9.5e-7	200	3.0e-1	5	1.8e-5	2.0
2^{-11}	2.4e-7	200	3.0e-1	5	4.6e-6	2.0
2^{-12}	6.0e-8	200	3.0e-1	5	1.1e-6	2.0
2^{-13}	1.5e-8	200	3.0e-1	5	2.8e-7	2.0
2^{-14}	3.7e-9	200	3.0e-1	5	7.1e-8	2.0

Semilinear elliptic equations

- PDE

$$\begin{cases} -\Delta u - (u-1)^3 + (u+2)^2 = f & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

  where $\Omega = (0,1)^d$.
- Example: $u(x) = 3\sin(2\pi x)$ for $d = 1$

Table: The performance of Int-Deep in 1D with different DNN width N and depth L when $h = \frac{1}{1024}$. T^{DL} denotes the running time of GPU for deep learning training in Pytorch.

L \ N	10				30				50			
	e_0^h	e^h	#K	T^{DL}	e_0^h	e^h	#K	T^{DL}	e_0^h	e^h	#K	T^{DL}
2	3.7e+0	1.8e-5	8	42.0s	2.3e-1	1.8e-5	5	42.0s	5.2e-2	1.8e-5	4	46.9s
4	1.1e-1	1.8e-5	4	48.1s	1.6e-2	1.8e-5	3	49.0s	1.5e-2	1.8e-5	3	48.7s
6	9.8e-2	1.8e-5	4	45.1s	8.5e-3	1.8e-5	3	50.4s	5.3e-3	1.8e-5	3	51.8s

Nonlinear Schrodinger Equation

$$\begin{cases} -\Delta u + V(x)u + 10u^3 = \lambda u & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

where $V(x)$ is two Gaussian wells, $\Omega = [0, 1]$.

Table: The performance of Int-Deep in 1D with different mesh sizes h.

h	#epoch	res^{DL}	e^{DL}	λ^{DL}	#K	res^{h}	e^{h}	λ^{h}
2^{-5}	500	6.6e+0	6.3e-2	24.3	4	4.6e-2	3.9e-3	25.1
2^{-6}	500	8.2e+0	6.4e-2	24.3	5	1.2e-2	3.3e-3	25.1
2^{-7}	500	17.5e+0	6.4e-2	24.3	6	3.0e-3	2.2e-3	25.1
2^{-8}	500	34.7e+0	6.4e-2	24.3	7	8.2e-4	1.1e-3	25.1
2^{-9}	500	50.2e+0	6.4e-2	24.3	7	4.2e-4	7.0e-4	25.1

Nonlinear Schrodinger Equation

$$\begin{cases} -\Delta u + V(x)u + 10u^3 = \lambda u & \text{in } \Omega, \\ u = 0 & \text{on } \partial\Omega, \end{cases}$$

where $V(x)$ is two Gaussian wells, $\Omega = [0, 1]$.

Table: The performance of Int-Deep in 1D with different DNN width N and depth L when $h = \frac{1}{512}$. T^{DL} denotes the running time of GPU for deep learning training in Pytorch.

L \ N	30				40				50			
	e^h	res^h	#K	T^{DL}	e^h	res^h	#K	T^{DL}	e^h	res^h	#K	T^{DL}
2	3.2e-2	2.7e-3	7	65.4s	2.1e+0	3.8e-3	7	64.9s	2.2e-3	9.2e-3	7	64.9s
4	2.1e+0	6.2e-3	7	68.0s	1.7e-1	3.8e-2	7	68.2s	7.8e-2	2.4e-3	7	68.5s
6	7.0e-2	3.4e-3	7	70.4s	5.2e-2	2.5e-3	7	69.5s	1.2e-3	3.0e-4	7	69.9s

Example 3: Identify Multiple Solutions

NND: Structure Probing Neural Network Deflation. Gu, Wang, and Y. (Submitted, 2020)

- Idea: leverage the power of NN representation for multiple solutions of nonlienar PDEs

Our philosophy

- Non-uniqueness of network representation.
- Many good local minimizers.

Conventional methods

- Functional deflation methods (Ferrell et al.)
- Numerical continuation or homopoty (Sommese and Morgan; Hao et al.; Liao)
- Davidenko differential equation (Branin; Davidenko)

Structure Probing Neural Network Deflation

Deflation

$$\min_{\boldsymbol{\theta}} \mathcal{L}_{\text{NND}} := \Big(\sum_{k=1}^{K} \frac{1}{\|u(\boldsymbol{x};\boldsymbol{\theta}) - u_k(\boldsymbol{x})\|_{L^2(\Omega)}^{p_k}} + \alpha\Big)\mathcal{L}(u(\boldsymbol{x};\boldsymbol{\theta})),$$

- $\mathcal{L}(u(\boldsymbol{x};\boldsymbol{\theta}))$ is the loss function in the least square method
- p_k is a positive power of the deflated solutions u_k
- $\alpha > 0$ is a shift constant to exclude spurious solutions

Structure Probing

$$u_J(\boldsymbol{x};\boldsymbol{\theta}_J) = u(\boldsymbol{x};\boldsymbol{\theta}) + \sum_{j=1}^{J} c_j \xi_j(\boldsymbol{x}),$$

- $\boldsymbol{\theta}_J := \{\boldsymbol{\theta}, \{c_j\}_{j=1}^{J}\}$ is trainable after initialization
- Structure probing functions $\xi_j(\boldsymbol{x})$
  - e.g., $\{\xi_j(\boldsymbol{x}) = e^{\mathrm{i}\boldsymbol{k}_j\cdot\boldsymbol{x}}, |\boldsymbol{k}_j| = j, j = 1, \cdots, J\}$ with $\boldsymbol{k}_j$ randomly selected
  - e.g., $\{\xi_j(\boldsymbol{x}) = \sin(j\pi|\boldsymbol{x}|), j = 1, \cdots, J\}$

We consider 2-D Yamabe's equation that seeks u such that

$$\begin{aligned} -8\Delta u - 0.1u + \frac{u^5}{|\boldsymbol{x}|^3} &= 0, \quad \text{in } \Omega = \{\boldsymbol{x} \in \mathbb{R}^2 : r < |\boldsymbol{x}| < R\}, \\ u &= 1, \quad \text{on } \partial\Omega, \end{aligned}$$

where r and R are set as 1 and 100.

- 9 solutions by using non-network deflation techniques and various initial guesses in (Farrell et al.)
- We got 14 solutions.

2-D Yamabe's equation

$$-8\Delta u - 0.1u + \frac{u^5}{|\boldsymbol{x}|^3} = 0, \quad \text{in } \Omega = \{\boldsymbol{x} \in \mathbb{R}^2 : r < |\boldsymbol{x}| < R\},$$

$$u = 1, \quad \text{on } \partial\Omega,$$

where r and R are set as 1 and 100.

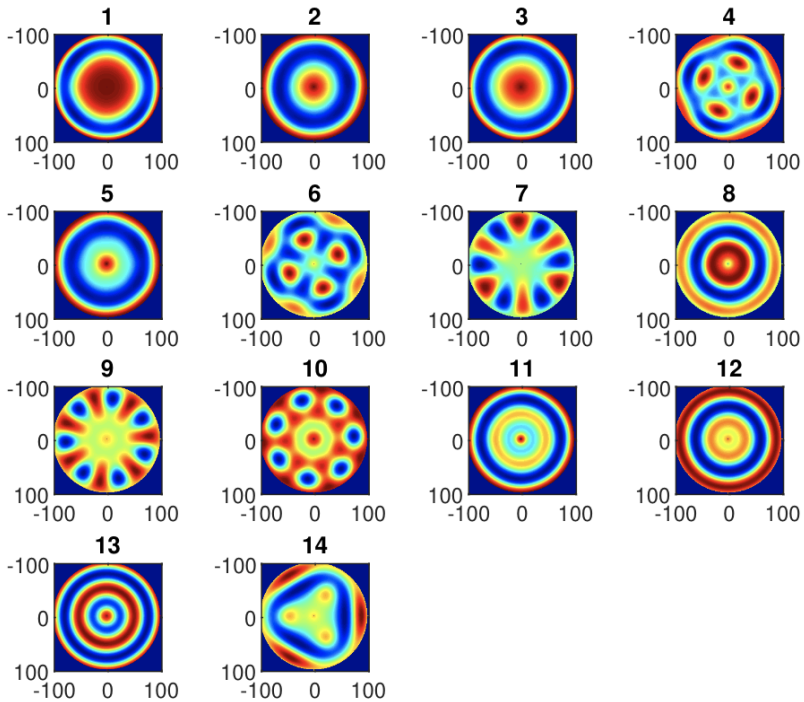


Figure: 14 solutions of the 2D problem.

The 6-dimensional Yamabe's equation

$$-\frac{4(d-1)}{(d-2)}\Delta u - 0.125u + \frac{u^{\frac{d+2}{(d-2)}}}{|\boldsymbol{x}|^3} = 0, \quad \text{in } \Omega = \{1 < |\boldsymbol{x}| < 100\},$$

$$u = 1, \quad \text{on } \partial\Omega,$$

where $d \geq 3$ is the dimension of the problem.

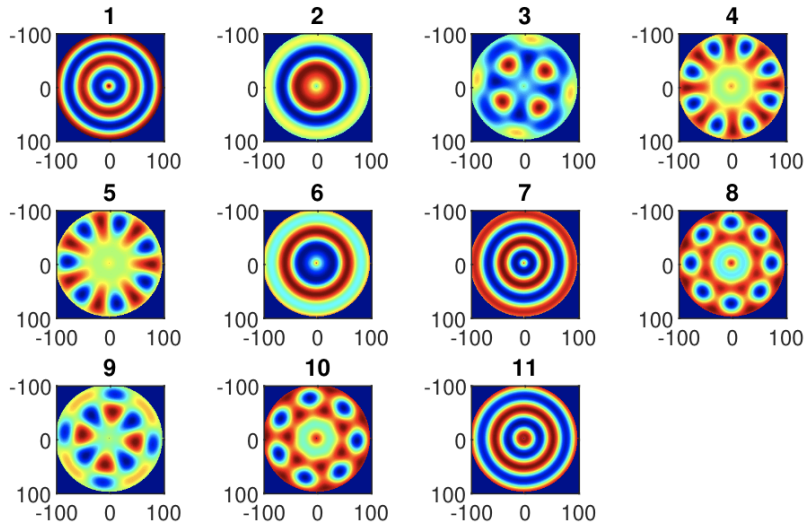


Figure: 11 solutions of the 6D problem.