A Discontinuity Capturing Shallow Neural Network for Anisotropic Elliptic Interface Problems

Wei-Fan Hu
wfhu@math.ncu.edu.tw

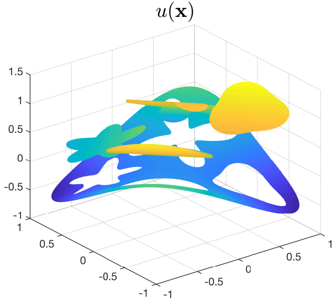
Department of Mathematics
National Central University
Taiwan

Anisotropic Elliptic Interface Problems

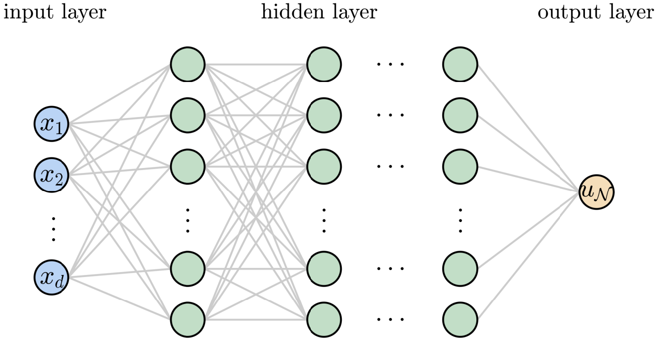
- The d-dimensional anisotropic elliptic interface problem is described by

$$\begin{cases} \nabla \cdot (A(\mathbf{x})\nabla u(\mathbf{x})) - \lambda(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}) & \text{in } \Omega = \bigcup_{\ell=0}^{L} \Omega_\ell \subset \mathbb{R}^d \\ [u] = v_\ell, \quad [A\nabla u \cdot \mathbf{n}] = w_\ell & \text{on } \Gamma_\ell \subset \mathbb{R}^{d-1} \text{ for } \ell = 1, 2, \cdots, L \\ u(\mathbf{x}) = g(\mathbf{x}) & \text{on } \partial\Omega \subset \mathbb{R}^{d-1} \end{cases}$$

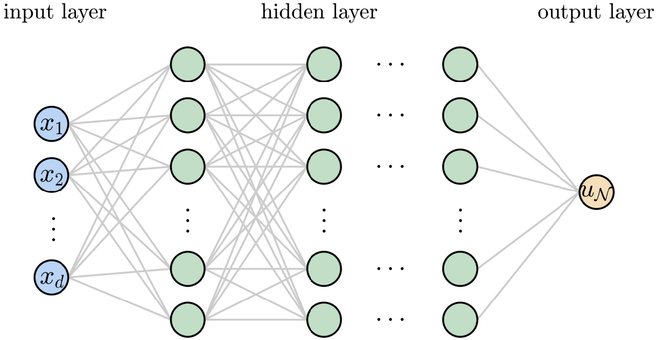
- $A(\mathbf{x}) \in \mathbb{R}^{d \times d}$ is symmetric positive definite and $\lambda > 0$
- $[\cdot]$ denotes the quantity of jump discontinuity
- Obviously, the solution u is *discontinuous* across all interfaces



L-Layer Deep Neural Network Architecture

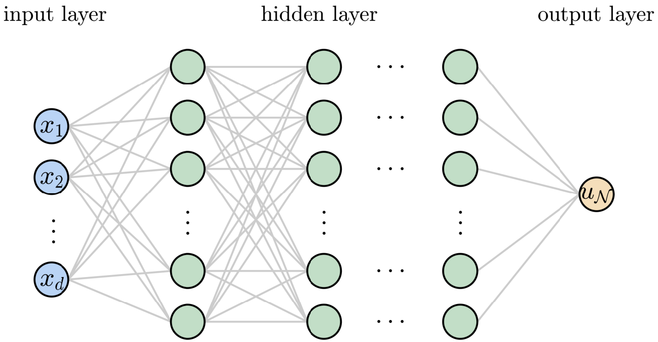


L-Layer Deep Neural Network Architecture



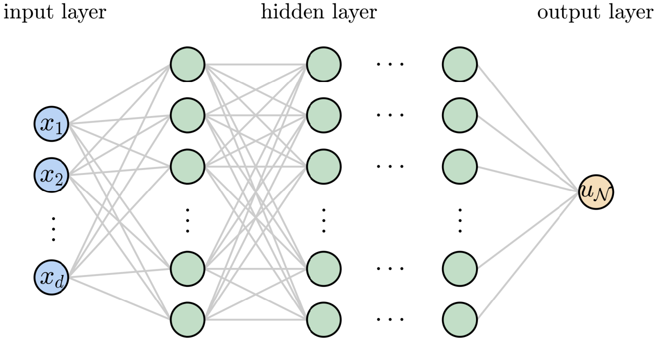
- Define $\mathbf{x}^{[0]} = \mathbf{x}$ and $\mathcal{N}_k(\mathbf{x}^{[k-1]}) = W^{[k]}\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]}$

L-Layer Deep Neural Network Architecture



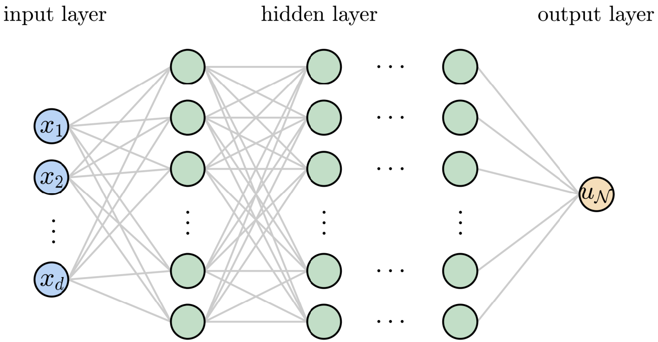
- Define $\mathbf{x}^{[0]} = \mathbf{x}$ and $\mathcal{N}_k(\mathbf{x}^{[k-1]}) = W^{[k]}\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]}$
- Neural network approximator $u_{\mathcal{N}}(\mathbf{x}) = \mathcal{N}_{L-1} \circ (\sigma \circ \mathcal{N}_{L-2}) \circ \cdots \circ (\sigma \circ \mathcal{N}_1)(\mathbf{x})$

L-Layer Deep Neural Network Architecture



- Define $\mathbf{x}^{[0]} = \mathbf{x}$ and $\mathcal{N}_k(\mathbf{x}^{[k-1]}) = W^{[k]}\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]}$
- Neural network approximator $u_{\mathcal{N}}(\mathbf{x}) = \mathcal{N}_{L-1} \circ (\sigma \circ \mathcal{N}_{L-2}) \circ \cdots \circ (\sigma \circ \mathcal{N}_1)(\mathbf{x})$
- $u_{\mathcal{N}}$ consists of composition of continuous functions, so $u_{\mathcal{N}}$ is *continuous*

L-Layer Deep Neural Network Architecture



- Define $\mathbf{x}^{[0]} = \mathbf{x}$ and $\mathcal{N}_k(\mathbf{x}^{[k-1]}) = W^{[k]}\mathbf{x}^{[k-1]} + \mathbf{b}^{[k]}$
- Neural network approximator $u_{\mathcal{N}}(\mathbf{x}) = \mathcal{N}_{L-1} \circ (\sigma \circ \mathcal{N}_{L-2}) \circ \cdots \circ (\sigma \circ \mathcal{N}_1)(\mathbf{x})$
- $u_{\mathcal{N}}$ consists of composition of continuous functions, so $u_{\mathcal{N}}$ is *continuous*

Question: How to approximate a discontinuous function using neural net approximation?

Continuous Function Extension

- Consider a d-dimensional, *piecewise continuous*, scalar function $u(\mathbf{x})$ in the domain $\Omega = \Omega^- \cup \Omega^+$ defined by

$$u(\mathbf{x}) = \begin{cases} u^-(\mathbf{x}) & \text{if } \mathbf{x} \in \Omega^- \\ u^+(\mathbf{x}) & \text{if } \mathbf{x} \in \Omega^+ \end{cases}$$

  where u^- and u^+ are both smooth functions in their corresponding subdomains

Continuous Function Extension

- Consider a d-dimensional, *piecewise continuous*, scalar function $u(\mathbf{x})$ in the domain $\Omega = \Omega^- \cup \Omega^+$ defined by

$$u(\mathbf{x}) = \begin{cases} u^-(\mathbf{x}) & \text{if } \mathbf{x} \in \Omega^- \\ u^+(\mathbf{x}) & \text{if } \mathbf{x} \in \Omega^+ \end{cases}$$

  where u^- and u^+ are both smooth functions in their corresponding subdomains

- Define the $(d+1)$-dimensional function using the augmentation variable $(\mathbf{x}, z)$ as

$$u_{\mathcal{N}}(\mathbf{x}, z) = \begin{cases} u^-(\mathbf{x}) & \text{if } z = -1 \\ u^+(\mathbf{x}) & \text{if } z = 1 \end{cases}$$

  where $\mathbf{x} \in \Omega$ and $z \in \mathbb{R}$

Continuous Function Extension

- Consider a d-dimensional, *piecewise continuous*, scalar function $u(\mathbf{x})$ in the domain $\Omega = \Omega^- \cup \Omega^+$ defined by

$$u(\mathbf{x}) = \begin{cases} u^-(\mathbf{x}) & \text{if } \mathbf{x} \in \Omega^- \\ u^+(\mathbf{x}) & \text{if } \mathbf{x} \in \Omega^+ \end{cases}$$

  where u^- and u^+ are both smooth functions in their corresponding subdomains

- Define the $(d+1)$-dimensional function using the augmentation variable $(\mathbf{x}, z)$ as

$$u_{\mathcal{N}}(\mathbf{x}, z) = \begin{cases} u^-(\mathbf{x}) & \text{if } z = -1 \\ u^+(\mathbf{x}) & \text{if } z = 1 \end{cases}$$

  where $\mathbf{x} \in \Omega$ and $z \in \mathbb{R}$

- Note that, both u^- and u^+ are now regarded as smooth extension over the entire domain Ω, so the augmented function $u_{\mathcal{N}}(\mathbf{x}, z)$ is assumed to be continuous on the domain $\Omega \times \mathbb{R}$

Continuous Function Extension

- Consider a d-dimensional, *piecewise continuous*, scalar function $u(\mathbf{x})$ in the domain $\Omega = \Omega^- \cup \Omega^+$ defined by

$$u(\mathbf{x}) = \begin{cases} u^-(\mathbf{x}) & \text{if } \mathbf{x} \in \Omega^- \\ u^+(\mathbf{x}) & \text{if } \mathbf{x} \in \Omega^+ \end{cases}$$

  where u^- and u^+ are both smooth functions in their corresponding subdomains

- Define the $(d+1)$-dimensional function using the augmentation variable $(\mathbf{x}, z)$ as

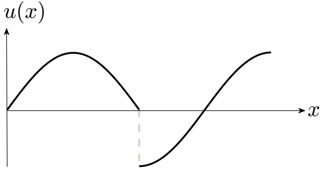
$$u_{\mathcal{N}}(\mathbf{x}, z) = \begin{cases} u^-(\mathbf{x}) & \text{if } z = -1 \\ u^+(\mathbf{x}) & \text{if } z = 1 \end{cases}$$

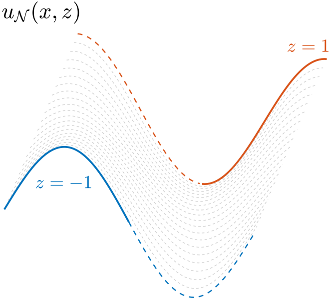
  where $\mathbf{x} \in \Omega$ and $z \in \mathbb{R}$

- Note that, both u^- and u^+ are now regarded as smooth extension over the entire domain Ω, so the augmented function $u_{\mathcal{N}}(\mathbf{x}, z)$ is assumed to be continuous on the domain $\Omega \times \mathbb{R}$

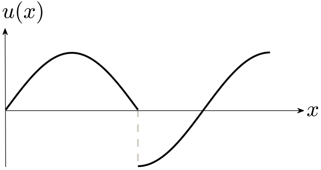
- u can be rewritten in terms of the augmented function as

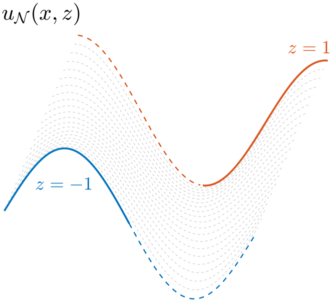
$$u(\mathbf{x}) = \begin{cases} u_{\mathcal{N}}(\mathbf{x}, -1) & \text{if } \mathbf{x} \in \Omega^- \\ u_{\mathcal{N}}(\mathbf{x}, 1) & \text{if } \mathbf{x} \in \Omega^+ \end{cases}$$



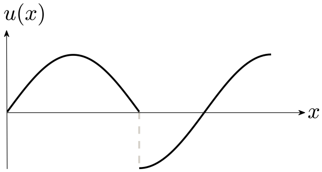


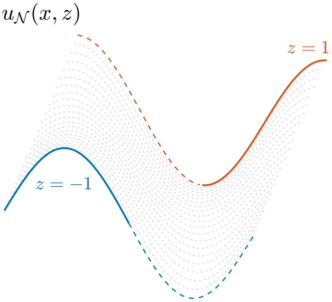
- Let $u(x) = \begin{cases} u^-(x) = \sin(2\pi x) & \text{if } x \in [0, \frac{1}{2}) \\ u^+(x) = \cos(2\pi x) & \text{if } x \in [\frac{1}{2}, 1] \end{cases}$



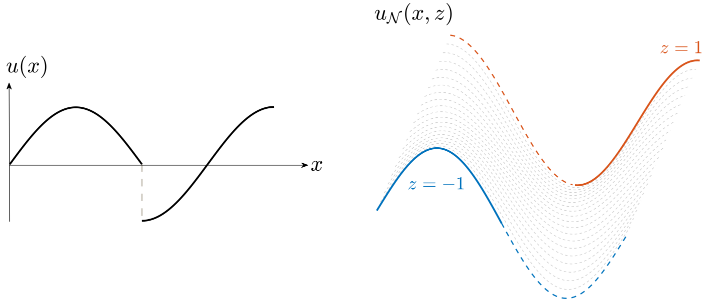


- Let $u(x) = \begin{cases} u^-(x) = \sin(2\pi x) & \text{if } x \in [0, \frac{1}{2}) \\ u^+(x) = \cos(2\pi x) & \text{if } x \in [\frac{1}{2}, 1] \end{cases}$
- We can simply find $u_{\mathcal{N}}(x, z) = \frac{1-z}{2} u^-(x) + \frac{1+z}{2} u^+(x)$

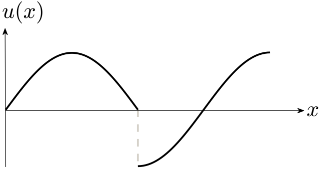


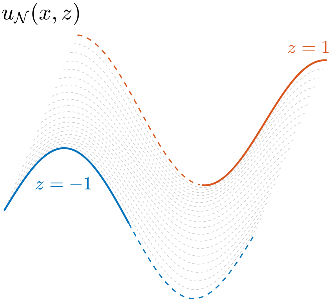


- Let $u(x) = \begin{cases} u^-(x) = \sin(2\pi x) & \text{if } x \in [0, \frac{1}{2}) \\ u^+(x) = \cos(2\pi x) & \text{if } x \in [\frac{1}{2}, 1] \end{cases}$
- We can simply find $u_{\mathcal{N}}(x, z) = \frac{1-z}{2} u^-(x) + \frac{1+z}{2} u^+(x)$
- There exists infinitely many such a function that has its restriction to be u



- Let $u(x) = \begin{cases} u^-(x) = \sin(2\pi x) & \text{if } x \in [0, \frac{1}{2}) \\ u^+(x) = \cos(2\pi x) & \text{if } x \in [\frac{1}{2}, 1] \end{cases}$
- We can simply find $u_{\mathcal{N}}(x,z) = \frac{1-z}{2}u^-(x) + \frac{1+z}{2}u^+(x)$
- There exists infinitely many such a function that has its restriction to be u
- Piecewise continuous functions with arbitrary many pieces can be done by simply labelling various z values

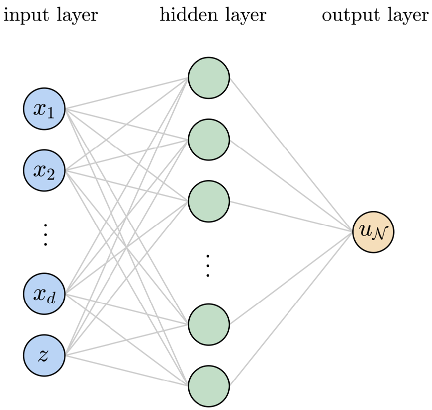




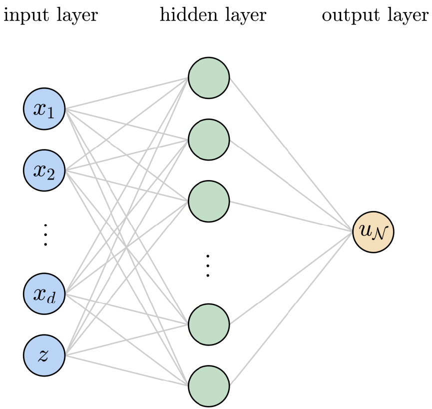
- Let $u(x) = \begin{cases} u^-(x) = \sin(2\pi x) & \text{if } x \in [0, \frac{1}{2}) \\ u^+(x) = \cos(2\pi x) & \text{if } x \in [\frac{1}{2}, 1] \end{cases}$
- We can simply find $u_{\mathcal{N}}(x, z) = \frac{1-z}{2} u^-(x) + \frac{1+z}{2} u^+(x)$
- There exists infinitely many such a function that has its restriction to be u
- Piecewise continuous functions with arbitrary many pieces can be done by simply labelling various z values

Remaining issue: How to construct the augmented function $u_{\mathcal{N}}$ using an approximation of neural network?

Discontinuity Capturing Shallow Neural Network

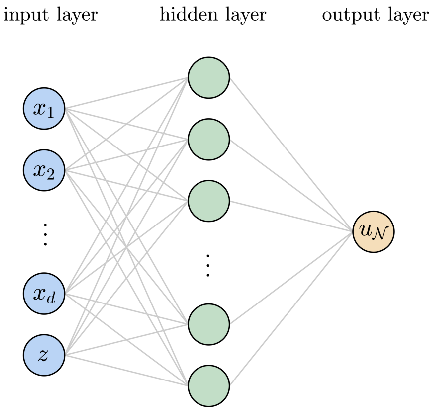


Discontinuity Capturing Shallow Neural Network



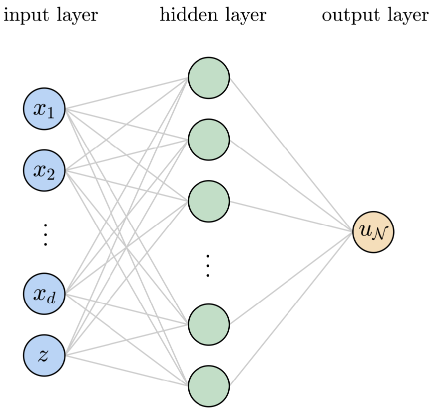
- DCSNN approximator $u_{\mathcal{N}}(\mathbf{x}, z) = W^{[2]}\sigma(W^{[1]}[\mathbf{x}, z] + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]}$

Discontinuity Capturing Shallow Neural Network



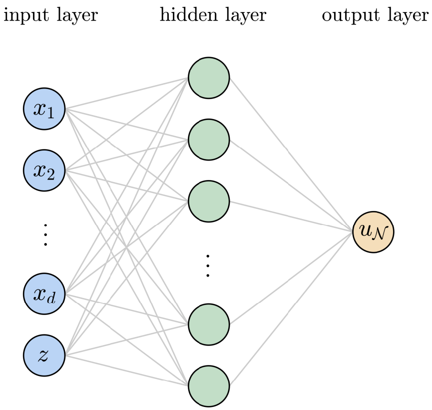
- DCSNN approximator $u_{\mathcal{N}}(\mathbf{x}, z) = W^{[2]}\sigma(W^{[1]}[\mathbf{x}, z] + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]}$
- N neurons are employed in the hidden layer

Discontinuity Capturing Shallow Neural Network



- DCSNN approximator $u_{\mathcal{N}}(\mathbf{x}, z) = W^{[2]}\sigma(W^{[1]}[\mathbf{x}, z] + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]}$
- N neurons are employed in the hidden layer
- Weight: $W^{[1]} \in \mathbb{R}^{N\times(d+1)}$, $W^{[2]} \in \mathbb{R}^{1\times N}$; bias: $\mathbf{b}^{[1]} \in \mathbb{R}^{N\times 1}$, $\mathbf{b}^{[2]} \in \mathbb{R}$

Discontinuity Capturing Shallow Neural Network



- DCSNN approximator $u_{\mathcal{N}}(\mathbf{x}, z) = W^{[2]}\sigma(W^{[1]}[\mathbf{x}, z] + \mathbf{b}^{[1]}) + \mathbf{b}^{[2]}$
- N neurons are employed in the hidden layer
- Weight: $W^{[1]} \in \mathbb{R}^{N\times(d+1)}$, $W^{[2]} \in \mathbb{R}^{1\times N}$; bias: $\mathbf{b}^{[1]} \in \mathbb{R}^{N\times 1}$, $\mathbf{b}^{[2]} \in \mathbb{R}$
- Total number of parameters $N_p = (d+3)N + 1$

Training Method: Levenberg-Marquardt Method

- Collecting all training parameters in the vector $\mathbf{p} \in \mathbb{R}^{N_p}$

Training Method: Levenberg-Marquardt Method

- Collecting all training parameters in the vector $\mathbf{p} \in \mathbb{R}^{N_p}$
- Given training points $\{(\mathbf{x}^i, z^i)\}_{i=1}^M$, where z^i is determined by identifying the category of $\mathbf{x}^i$, and target outputs $\{u(\mathbf{x}^i)\}_{i=1}^M$

Training Method: Levenberg-Marquardt Method

- Collecting all training parameters in the vector $\mathbf{p} \in \mathbb{R}^{N_p}$
- Given training points $\{(\mathbf{x}^i, z^i)\}_{i=1}^M$, where z^i is determined by identifying the category of $\mathbf{x}^i$, and target outputs $\{u(\mathbf{x}^i)\}_{i=1}^M$
- All training parameters can be learned via minimizing the mean squared error

$$\text{Loss}(\mathbf{p}) = \frac{1}{M}\sum_{i=1}^{M}\left(u(\mathbf{x}^i) - u_{\mathcal{N}}(\mathbf{x}^i, z^i; \mathbf{p})\right)^2$$

Training Method: Levenberg-Marquardt Method

- Collecting all training parameters in the vector $\mathbf{p} \in \mathbb{R}^{N_p}$
- Given training points $\{(\mathbf{x}^i, z^i)\}_{i=1}^M$, where z^i is determined by identifying the category of $\mathbf{x}^i$, and target outputs $\{u(\mathbf{x}^i)\}_{i=1}^M$
- All training parameters can be learned via minimizing the mean squared error

$$\text{Loss}(\mathbf{p}) = \frac{1}{M} \sum_{i=1}^{M} \left(u(\mathbf{x}^i) - u_{\mathcal{N}}(\mathbf{x}^i, z^i; \mathbf{p}) \right)^2$$

- Levenberg-Marquardt method

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \left(J^T J + \mu I \right)^{-1} \underbrace{\left[J^T (\mathbf{u} - \mathbf{u}_{\mathcal{N}}(\mathbf{p}^{(k)})) \right]}_{-\frac{1}{2} \nabla \text{Loss}(\mathbf{p}^{(k)})}$$

- Jacobian matrix $J = \partial \mathbf{u}_{\mathcal{N}} / \partial \mathbf{p} \in \mathbb{R}^{M \times N_p}$ (typically $M > N_p$); the computation of J can be done using *auto differentiation*

Training Method: Levenberg-Marquardt Method

- Collecting all training parameters in the vector $\mathbf{p} \in \mathbb{R}^{N_p}$
- Given training points $\{(\mathbf{x}^i, z^i)\}_{i=1}^{M}$, where z^i is determined by identifying the category of $\mathbf{x}^i$, and target outputs $\{u(\mathbf{x}^i)\}_{i=1}^{M}$
- All training parameters can be learned via minimizing the mean squared error

$$\text{Loss}(\mathbf{p}) = \frac{1}{M}\sum_{i=1}^{M}\left(u(\mathbf{x}^i) - u_{\mathcal{N}}(\mathbf{x}^i, z^i; \mathbf{p})\right)^2$$

- Levenberg-Marquardt method

$$\mathbf{p}^{(k+1)} = \mathbf{p}^{(k)} + \left(J^T J + \mu I\right)^{-1} \underbrace{\left[J^T(\mathbf{u} - \mathbf{u}_{\mathcal{N}}(\mathbf{p}^{(k)}))\right]}_{-\frac{1}{2}\nabla \text{Loss}(\mathbf{p}^{(k)})}$$

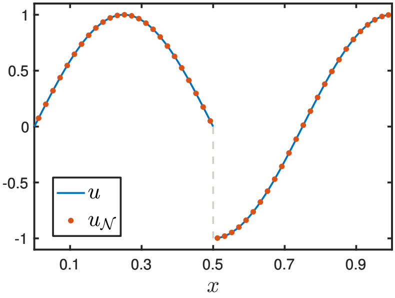
- Jacobian matrix $J = \partial \mathbf{u}_{\mathcal{N}} / \partial \mathbf{p} \in \mathbb{R}^{M \times N_p}$ (typically $M > N_p$); the computation of J can be done using *auto differentiation*
- The linear system (the second term) in each iteration is solved using *Singular Value Decomposition* or *Cholesky factorization*

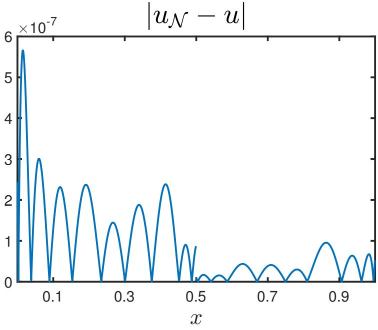
Testing Example

- The 1D target function is given by $u(x) = \begin{cases} \sin(2\pi x) & \text{if } x \in [0, \frac{1}{2}) \\ \cos(2\pi x) & \text{if } x \in [\frac{1}{2}, 1] \end{cases}$
- Only $N = 5$ neurons are used in the hidden layer, thus the total number of parameters $N_p = 21$
- 100 randomly sampled training points in the interval $[0, 1]$
- Sigmoid activation function
- Terminate the training iteration when $\text{Loss}(\mathbf{p}) < 10^{-12}$

Testing Example

- The 1D target function is given by $u(x) = \begin{cases} \sin(2\pi x) & \text{if } x \in [0, \frac{1}{2}) \\ \cos(2\pi x) & \text{if } x \in [\frac{1}{2}, 1] \end{cases}$
- Only $N = 5$ neurons are used in the hidden layer, thus the total number of parameters $N_p = 21$
- 100 randomly sampled training points in the interval $[0, 1]$
- Sigmoid activation function
- Terminate the training iteration when $\text{Loss}(\mathbf{p}) < 10^{-12}$





Theorem (Meer et al. 2021)

Consider the well-posed PDE of order k given by

$$\begin{cases} \mathcal{L}(u) = f & \text{in the domain } \Omega, \\ \mathcal{B}(u) = g & \text{on the boundary } \partial\Omega. \end{cases}$$

Let the exact solution of this PDE be given by u and let the loss functional be given by

$$Loss(\hat{u}) = \frac{1}{|\Omega|} \int_{\Omega} |\mathcal{L}(\hat{u}) - f|^2 \, \mathrm{d}\mathbf{x} + \frac{1}{|\partial\Omega|} \int_{\partial\Omega} |\mathcal{B}(\hat{u}) - g|^2 \, \mathrm{d}\mathbf{x}.$$

Consider some approximate solution $\hat{u}$ of which the first k (partial) derivatives exist and have finite L_2 norm. Then, for any $\varepsilon > 0$ there exists a $\delta(\varepsilon) > 0$ such that

$$Loss(\hat{u}) < \delta \implies \|\hat{u} - u\| < \varepsilon.$$

Physics-Informed Learning Machine

- Recall

$$\begin{cases} \nabla \cdot (A(\mathbf{x}) \nabla u(\mathbf{x})) - \lambda(\mathbf{x}) u(\mathbf{x}) = f(\mathbf{x}) & \text{in } \Omega = \bigcup_{\ell=0}^{L} \Omega_\ell \subset \mathbb{R}^d \\ [u] = v_\ell, \quad [A \nabla u \cdot \mathbf{n}] = w_\ell & \text{on } \Gamma_\ell \subset \mathbb{R}^{d-1} \text{ for } \ell = 1, 2, \cdots, L \\ u(\mathbf{x}) = g(\mathbf{x}) & \text{on } \partial\Omega \subset \mathbb{R}^{d-1} \end{cases}$$

Physics-Informed Learning Machine

- Recall

$$\begin{cases} \nabla \cdot (A(\mathbf{x})\nabla u(\mathbf{x})) - \lambda(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}) & \text{in } \Omega = \bigcup_{\ell=0}^{L} \Omega_\ell \subset \mathbb{R}^d \\ [u] = v_\ell, \quad [A\nabla u \cdot \mathbf{n}] = w_\ell & \text{on } \Gamma_\ell \subset \mathbb{R}^{d-1} \text{ for } \ell = 1, 2, \cdots, L \\ u(\mathbf{x}) = g(\mathbf{x}) & \text{on } \partial\Omega \subset \mathbb{R}^{d-1} \end{cases}$$

- $[u_{\mathcal{N}}] = u_{\mathcal{N}}(\mathbf{x}, z_0) - u_{\mathcal{N}}(\mathbf{x}, z_\ell)$ for $\mathbf{x} \in \Gamma$; the same manner applies for $[A\nabla_{\mathbf{x}} u_{\mathcal{N}} \cdot \mathbf{n}]$

Physics-Informed Learning Machine

- Recall

$$\begin{cases} \nabla\cdot(A(\mathbf{x})\nabla u(\mathbf{x})) - \lambda(\mathbf{x})u(\mathbf{x}) = f(\mathbf{x}) & \text{in } \Omega = \bigcup_{\ell=0}^{L}\Omega_\ell \subset \mathbb{R}^d \\ [u] = v_\ell, \quad [A\nabla u\cdot\mathbf{n}] = w_\ell & \text{on } \Gamma_\ell \subset \mathbb{R}^{d-1} \text{ for } \ell = 1, 2, \cdots, L \\ u(\mathbf{x}) = g(\mathbf{x}) & \text{on } \partial\Omega \subset \mathbb{R}^{d-1} \end{cases}$$

- $[u_{\mathcal{N}}] = u_{\mathcal{N}}(\mathbf{x}, z_0) - u_{\mathcal{N}}(\mathbf{x}, z_\ell)$ for $\mathbf{x} \in \Gamma$; the same manner applies for $[A\nabla_{\mathbf{x}} u_{\mathcal{N}} \cdot \mathbf{n}]$
- Given training points $\{(\mathbf{x}^i, z^i)\}_{i=1}^{M}$ in Ω, $\{\mathbf{x}^j_{\partial\Omega}\}_{j=1}^{M_b}$ on $\partial\Omega$, and $\{\mathbf{x}^k_{\Gamma_\ell}\}_{k=1}^{M_{\Gamma_\ell}}$ along Γ_ℓ

Physics-Informed Learning Machine

- Recall

$$\begin{cases} \nabla\cdot(A(\mathbf{x})\nabla u(\mathbf{x}))-\lambda(\mathbf{x})u(\mathbf{x})=f(\mathbf{x}) & \text{in } \Omega=\bigcup_{\ell=0}^{L}\Omega_\ell\subset\mathbb{R}^d \\ [u]=v_\ell,\quad [A\nabla u\cdot\mathbf{n}]=w_\ell & \text{on } \Gamma_\ell\subset\mathbb{R}^{d-1} \text{ for } \ell=1,2,\cdots,L \\ u(\mathbf{x})=g(\mathbf{x}) & \text{on } \partial\Omega\subset\mathbb{R}^{d-1} \end{cases}$$

- $[u_{\mathcal{N}}]=u_{\mathcal{N}}(\mathbf{x},z_0)-u_{\mathcal{N}}(\mathbf{x},z_\ell)$ for $\mathbf{x}\in\Gamma$; the same manner applies for $[A\nabla_{\mathbf{x}}u_{\mathcal{N}}\cdot\mathbf{n}]$
- Given training points $\{(\mathbf{x}^i,z^i)\}_{i=1}^{M}$ in Ω, $\{\mathbf{x}^j_{\partial\Omega}\}_{j=1}^{M_b}$ on $\partial\Omega$, and $\{\mathbf{x}^k_{\Gamma_\ell}\}_{k=1}^{M_{\Gamma_\ell}}$ along Γ_ℓ
- Solving the differential equation is converted to the optimization problem

$$\begin{aligned} \text{Loss}(\mathbf{p}) &= \frac{1}{M}\sum_{i=1}^{M}\left[\nabla_{\mathbf{x}}\cdot(A(\mathbf{x}^i)\nabla_{\mathbf{x}}u_{\mathcal{N}}(\mathbf{x}^i,z^i))-\lambda(\mathbf{x}^i)u_{\mathcal{N}}(\mathbf{x}^i,z^i)-f(\mathbf{x}^i)\right]^2 \\ &+\frac{1}{M_b}\sum_{j=1}^{M_b}\left[u_{\mathcal{N}}(\mathbf{x}^j_{\partial\Omega},z_0)-g(\mathbf{x}^j_{\partial\Omega})\right]^2 \\ &+\sum_{\ell=1}^{L}\frac{1}{M_{\Gamma_\ell}}\left(\sum_{k=1}^{M_{\Gamma_\ell}}\left([u_{\mathcal{N}}]-v_\ell(\mathbf{x}^k_{\Gamma_\ell})\right)^2+\left([A\nabla_{\mathbf{x}}u_{\mathcal{N}}\cdot\mathbf{n}]-w_\ell(\mathbf{x}^k_{\Gamma_\ell})\right)^2\right) \end{aligned}$$

Example 1: 2D Problem with Regular Domain

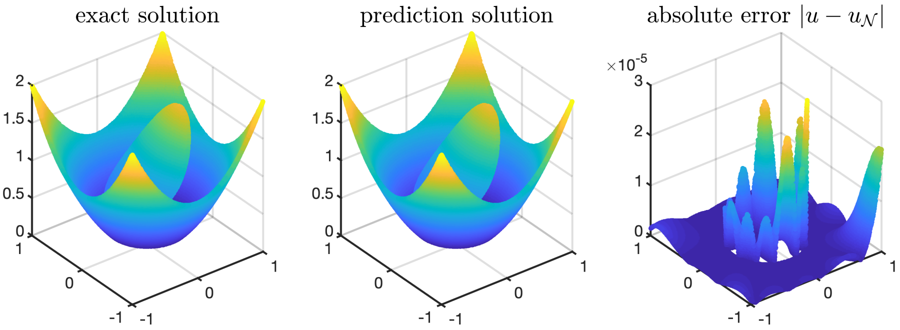
- Domain $\Omega = [-1,1] \times [-1,1]$ and interface $\Gamma : \left(\frac{x_1}{0.5}\right)^2 + \left(\frac{x_2}{0.5}\right)^2 = 1$
- We set

$$u(x_1,x_2) = \begin{cases} u_0 = x_1^2 + x_2^2 & \text{if } (x_1,x_2) \in \Omega_0 \\ u_1 = \exp(x_1)\cos(x_2) & \text{if } (x_1,x_2) \in \Omega_1 \end{cases}$$

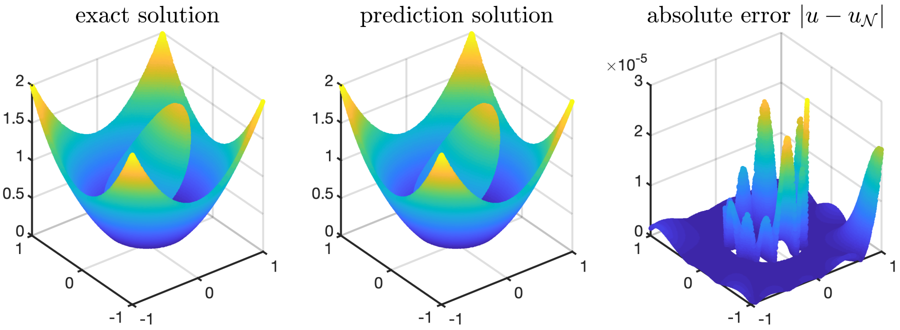
$$A(x_1,x_2) = \begin{cases} A_0 = 1000 \begin{bmatrix} x_1^2 + x_2^2 + 1 & x_1^2 + x_2^2 \\ x_1^2 + x_2^2 & x_1^2 + x_2^2 + 2 \end{bmatrix} & \text{if } (x_1,x_2) \in \Omega_0, \\ A_1 = \frac{1}{1000} A_0 & \text{if } (x_1,x_2) \in \Omega_1, \end{cases}$$

$$\lambda(x_1,x_2) = \begin{cases} \lambda_0 = 1000 \exp(x_1)(x_1^2 + x_2^2 + 3)\sin(x_2) & \text{if } (x_1,x_2) \in \Omega_0, \\ \lambda_1 = \frac{1}{1000}\lambda_0 & \text{if } (x_1,x_2) \in \Omega_1. \end{cases}$$

- $M = 225$ interior points in the computational domain Ω
  $M_b = 60$ points on the boundary $\partial\Omega$
  $M_\Gamma = 60$ points on the interface Γ



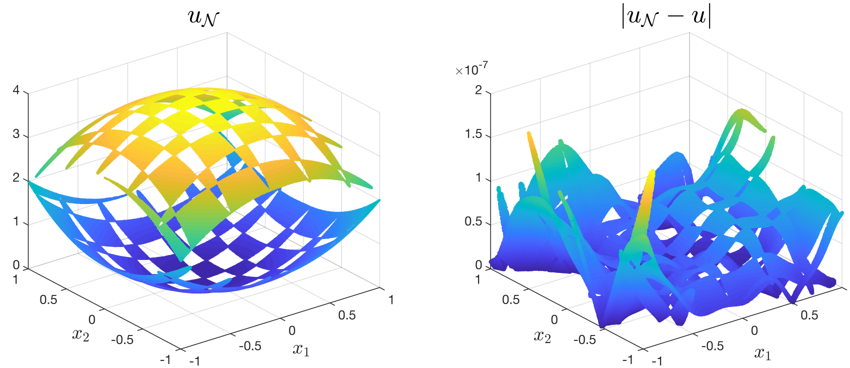
exact solution
prediction solution
absolute error $|u - u_{\mathcal{N}}|$
$\times 10^{-5}$



N_{deg}	$\|u_{IIM} - u\|_\infty$	(N, N_p)	$\|u_{\mathcal{N}} - u\|_\infty$	$\|u_{\mathcal{N}} - u\|_2$
65536	8.008E−05	(30, 150)	5.259E−05	9.038E−06
262144	2.091E−05	(40, 200)	1.661E−05	2.352E−06

Table: u: Exact solution. u_{IIM}: Solution obtained by IIM. N_{deg} = 65536 and 262144 correspond to $m = 256$ and $m = 512$. $u_{\mathcal{N}}$: Solution obtained from DCSNN model.

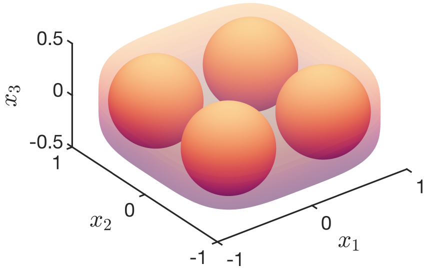
Example 2: 2D Problem with complicated geometry



N_{deg}	$\|u_{FEM} - u\|_\infty$	$\|\nabla u_{FEM} - \nabla u\|_\infty$	(N, N_p)	$\|u_{\mathcal{N}} - u\|_\infty$	$\|\nabla u_{\mathcal{N}} - \nabla u\|_\infty$
25600	9.400E−05	1.433E−03	(10, 50)	3.490E−06	6.087E−06
102400	2.600E−05	6.890E−04	(20, 100)	1.998E−07	6.318E−07

Table: u: Exact solution. u_{FEM}: Solution obtained by FEM. $N_{deg} = 25600$ and 102400 correspond to $m = 160$ and $m = 320$. $u_{\mathcal{N}}$: Solution obtained from DCSNN model.

Example 3: 3D Problem



- The exact solution is chosen as

$$u(x_1,x_2,x_3)=\begin{cases} u_0=\exp(x_1+x_2+x_3) & \text{if } (x_1,x_2,x_3)\in\Omega_0,\\ u_1=\sin x_1 \sin x_2 \sin x_3 & \text{if } (x_1,x_2,x_3)\in\Omega_1,\\ u_2=\cos x_1 \cos x_2 \cos x_3 & \text{if } (x_1,x_2,x_3)\in\Omega_2,\\ u_3=\cosh x_1 \cosh x_2 \cosh x_3 & \text{if } (x_1,x_2,x_3)\in\Omega_3,\\ u_4=\sinh x_1 \sinh x_2 \sinh x_3 & \text{if } (x_1,x_2,x_3)\in\Omega_4. \end{cases}$$

(N, N_p)	$\|u_{\mathcal{N}}-u\|_\infty$	$\|u_{\mathcal{N}}-u\|_2$
(40, 240)	2.337E−04	3.696E−05
(50, 300)	1.951E−05	4.715E−06

References

1. W.-F. Hu, T.-S. Lin, and M.-C. Lai
   A discontinuity capturing shallow neural network for elliptic interface problems
   arXiv: 2106.05587
2. W.-F. Hu, T.-S. Lin, and M.-C. Lai
   Solving anisotropic elliptic interface problems by machine learning
   in preparation

Thank you for your attention!