Finite Element Operator Network: Theory and Computation

Seungchan Ko

Department of Mathematics
Inha University

November 8, 2024

KIAS AI and Natural Sciences 2024 Fall Workshop

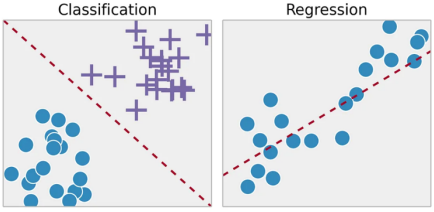
What is deep learning?

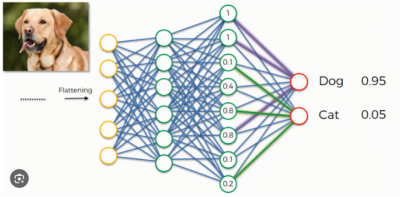
What is deep learning?

- Supervised learning problem:
  - Given data $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $\boldsymbol{x}_i \sim \mu^*$ and $y_i = f^*(\boldsymbol{x}_i) + \varepsilon_i$.
  - Learn the relation between $\boldsymbol{x}$ and y (the target function f^*) from $\mathcal{S}$.

What is deep learning?

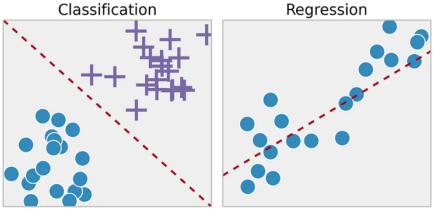
- Supervised learning problem:
  - Given data $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $\boldsymbol{x}_i \sim \mu^*$ and $y_i = f^*(\boldsymbol{x}_i) + \varepsilon_i$.
  - Learn the relation between $\boldsymbol{x}$ and y (the target function f^*) from $\mathcal{S}$.

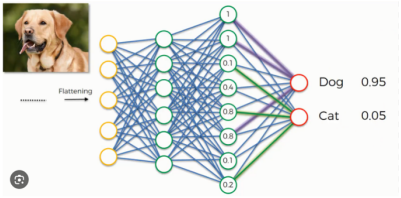




What is deep learning?

- Supervised learning problem:
  - Given data $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $\boldsymbol{x}_i \sim \mu^*$ and $y_i = f^*(\boldsymbol{x}_i) + \varepsilon_i$.
  - Learn the relation between $\boldsymbol{x}$ and y (the target function f^*) from $\mathcal{S}$.

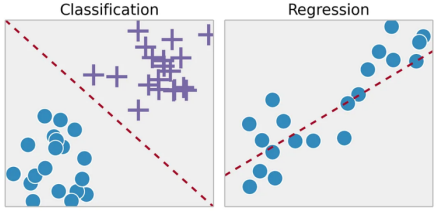


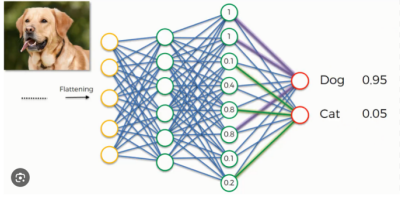


- Training a neural network on data approximates the underlying mapping function from inputs to outputs.

What is deep learning?

- Supervised learning problem:
  - Given data $\mathcal{S} = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, $\boldsymbol{x}_i \sim \mu^*$ and $y_i = f^*(\boldsymbol{x}_i) + \varepsilon_i$.
  - Learn the relation between $\boldsymbol{x}$ and y (the target function f^*) from $\mathcal{S}$.





- Training a neural network on data approximates the underlying mapping function from inputs to outputs.
- We say "*approximate*" because although we suspect such a mapping function exists, we don't know anything about it.

Deep Learning

- **Viewpoint**: Deep learning as a function approximation.

Deep Learning

- **Viewpoint**: Deep learning as a function approximation.
- **Objective of this talk**: Mathematical analysis of deep learning within the numerical analysis framework.

Deep Learning

- **Viewpoint**: Deep learning as a function approximation.
- **Objective of this talk**: Mathematical analysis of deep learning within the numerical analysis framework.

  "Good" regularity of target function $\implies$ "Good" approximation

Deep Learning

- **Viewpoint**: Deep learning as a function approximation.
- **Objective of this talk**: Mathematical analysis of deep learning within the numerical analysis framework.

"Good" regularity of target function $\Longrightarrow$ "Good" approximation

(Ex 1) Taylor's theorem: C^k regularity $\longrightarrow$ approximation with polynomial of degree k.

Deep Learning

- **Viewpoint**: Deep learning as a function approximation.

- **Objective of this talk**: Mathematical analysis of deep learning within the numerical analysis framework.

"Good" regularity of target function $\implies$ "Good" approximation

(Ex 1) Taylor's theorem: C^k regularity $\longrightarrow$ approximation with polynomial of degree k.

(Ex 2) 1D Trapezoidal rule: C^2 regularity $\longrightarrow$ error of order $\mathcal{O}(m^{-2})$.

- **Viewpoint**: Deep learning as a function approximation.

- **Objective of this talk**: Mathematical analysis of deep learning within the numerical analysis framework.

  "Good" regularity of target function $\Longrightarrow$ "Good" approximation

(Ex 1) Taylor's theorem: C^k regularity $\longrightarrow$ approximation with polynomial of degree k.

(Ex 2) 1D Trapezoidal rule: C^2 regularity $\longrightarrow$ error of order $\mathcal{O}(m^{-2})$.

(Ex 3) A priori estimate for FEM: $\|u - u_h\|_{H^1(\Omega)} \leq Ch^{m-1}|u|_{H^m(\Omega)}$.

- **Main questions of interests**

- **Viewpoint**: Deep learning as a function approximation.

- **Objective of this talk**: Mathematical analysis of deep learning within the numerical analysis framework.

  "Good" regularity of target function $\Longrightarrow$ "Good" approximation

(Ex 1) Taylor's theorem: C^k regularity $\longrightarrow$ approximation with polynomial of degree k.

(Ex 2) 1D Trapezoidal rule: C^2 regularity $\longrightarrow$ error of order $\mathcal{O}(m^{-2})$.

(Ex 3) A priori estimate for FEM: $\|u - u_h\|_{H^1(\Omega)} \leq Ch^{m-1}|u|_{H^m(\Omega)}$.

- **Main questions of interests**
  - What is the proper regularity for the neural-network approximation?

- **Viewpoint**: Deep learning as a function approximation.

- **Objective of this talk**: Mathematical analysis of deep learning within the numerical analysis framework.

  "Good" regularity of target function $\Longrightarrow$ "Good" approximation

(Ex 1) Taylor's theorem: C^k regularity $\longrightarrow$ approximation with polynomial of degree k.

(Ex 2) 1D Trapezoidal rule: C^2 regularity $\longrightarrow$ error of order $\mathcal{O}(m^{-2})$.

(Ex 3) A priori estimate for FEM: $\|u - u_h\|_{H^1(\Omega)} \leq Ch^{m-1}|u|_{H^m(\Omega)}$.

- **Main questions of interests**
  - What is the proper regularity for the neural-network approximation?
  - What is the proper mathematical framework for the analysis of neural-network approximation?

- **Viewpoint**: Deep learning as a function approximation.

- **Objective of this talk**: Mathematical analysis of deep learning within the numerical analysis framework.

  "Good" regularity of target function $\Longrightarrow$ "Good" approximation

(Ex 1) Taylor's theorem: C^k regularity $\longrightarrow$ approximation with polynomial of degree k.

(Ex 2) 1D Trapezoidal rule: C^2 regularity $\longrightarrow$ error of order $\mathcal{O}(m^{-2})$.

(Ex 3) A priori estimate for FEM: $\|u - u_h\|_{H^1(\Omega)} \leq Ch^{m-1}|u|_{H^m(\Omega)}$.

- **Main questions of interests**
  - What is the proper regularity for the neural-network approximation?
  - What is the proper mathematical framework for the analysis of neural-network approximation?
  - In this framework, how can we analyze errors arise from the neural-network approximation?

Polynomial Approximation

Polynomial Approximation

1. For a given target function $f(x)$, we aim to approximate it as a polynomial centered at x_0 and parametrized by the coefficients $a_0, a_1, \cdots, a_n$.

$$f(x) \approx a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + a_3(x - x_0)^3 + \cdots + a_n(x - x_0)^n.$$

Polynomial Approximation

1. For a given target function $f(x)$, we aim to approximate it as a polynomial centered at x_0 and parametrized by the coefficients $a_0, a_1, \cdots, a_n$.

$$f(x) \approx a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + a_3(x - x_0)^3 + \cdots + a_n(x - x_0)^n.$$

2. How to determine the parameters (coefficients) which best approximate the target function?

Polynomial Approximation

1. For a given target function $f(x)$, we aim to approximate it as a polynomial centered at x_0 and parametrized by the coefficients $a_0, a_1, \cdots, a_n$.

$$f(x) \approx a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + a_3(x - x_0)^3 + \cdots + a_n(x - x_0)^n.$$

2. How to determine the parameters (coefficients) which best approximate the target function?

$$a_k = \frac{f^{(k)}(x_0)}{k!}.$$

Polynomial Approximation

1. For a given target function $f(x)$, we aim to approximate it as a polynomial centered at x_0 and parametrized by the coefficients $a_0, a_1, \cdots, a_n$.

$$f(x) \approx a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + a_3(x - x_0)^3 + \cdots + a_n(x - x_0)^n.$$

2. How to determine the parameters (coefficients) which best approximate the target function?

$$a_k = \frac{f^{(k)}(x_0)}{k!}.$$

3. Is this a reliable approximation?

Polynomial Approximation

1. For a given target function $f(x)$, we aim to approximate it as a polynomial centered at x_0 and parametrized by the coefficients $a_0, a_1, \cdots, a_n$.

$$f(x) \approx a_0 + a_1(x - x_0) + a_2(x - x_0)^2 + a_3(x - x_0)^3 + \cdots + a_n(x - x_0)^n.$$

2. How to determine the parameters (coefficients) which best approximate the target function?

$$a_k = \frac{f^{(k)}(x_0)}{k!}.$$

3. Is this a reliable approximation?

   Convergence of the Taylor Series

Neural-Network Approximation

Neural-Network Approximation

1. For a given target function $f(x)$, we aim to approximate it as an L-layer feed-forward neural network parametrized by θ (weight and bias).

$$
\begin{aligned}
&f^1(x) = W^1 x + b^1, \\
&f^\ell(x) = W^\ell \sigma(f^{\ell-1}(x)) + b^\ell \quad \text{for } 2 \le \ell \le L \qquad \Longrightarrow \qquad f(x) \approx f^L(x)
\end{aligned}
$$

Neural-Network Approximation

1. For a given target function $f(x)$, we aim to approximate it as an L-layer feed-forward neural network parametrized by θ (weight and bias).

$$
\begin{aligned}
f^1(x) &= W^1 x + b^1, \\
f^\ell(x) &= W^\ell \sigma(f^{\ell-1}(x)) + b^\ell \quad \text{for } 2 \le \ell \le L \qquad \Longrightarrow \qquad f(x) \approx f^L(x)
\end{aligned}
$$

2. How to determine the parameters (weight, bias) which best approximate the target function?

   Set a loss function and minimize it. ('learning' as well as 'approximation')

Neural-Network Approximation

1. For a given target function $f(x)$, we aim to approximate it as an L-layer feed-forward neural network parametrized by θ (weight and bias).

$$
\begin{aligned}
f^1(x) &= W^1 x + b^1, \\
f^\ell(x) &= W^\ell \sigma(f^{\ell-1}(x)) + b^\ell \quad \text{for } 2 \le \ell \le L \qquad \Longrightarrow \qquad f(x) \approx f^L(x)
\end{aligned}
$$

2. How to determine the parameters (weight, bias) which best approximate the target function?

   Set a loss function and minimize it. ('learning' as well as 'approximation')

3. Is this a reliable approximation?

Neural-Network Approximation

1. For a given target function $f(x)$, we aim to approximate it as an L-layer feed-forward neural network parametrized by θ (weight and bias).

$$
\begin{aligned}
f^1(x) &= W^1 x + b^1, \\
f^\ell(x) &= W^\ell \sigma(f^{\ell-1}(x)) + b^\ell \quad \text{for } 2 \le \ell \le L \qquad \Longrightarrow \qquad f(x) \approx f^L(x)
\end{aligned}
$$

2. How to determine the parameters (weight, bias) which best approximate the target function?

   Set a loss function and minimize it. ('learning' as well as 'approximation')

3. Is this a reliable approximation?

   Universal approximation theorem

Universal Approximation Theorem

Let K be a compact subset of $\mathbb{R}^d$ and $f \in C(K, \mathbb{R}^D)$. Then we have

$$\lim_{n \to \infty} \inf_{g \in \mathcal{N}_n} \|g - f\|_{C(K)} = 0.$$

- Arbitrary width case for two-layer neural networks ($L = 2$, $n_1 \to \infty$).



G. Cybenko,
Approximation by superpositions of a sigmoidal function,
Math. Control Signals Systems, 2(4):303–314, 1989.



K. Hornik,
Approximation capabilities of multilayer feedforward networks,
Neural Networks, 4(2):251–257, 1991.

- Bounded width and arbitrary depth case ($n_i < \infty$, $L \to \infty$).



P. Kidger and T. Lyons,
Universal Approximation with Deep Narrow Network,
Proceedings of Machine Learning Research, 2306–2327, 2020.

Neural-Network Approximation

- Is the space of continuous function enough for the neural-network approximation?

Neural-Network Approximation

- Is the space of continuous function enough for the neural-network approximation?
- No! The Curse of Dimensionality:

$$\|f_m - f\|_{L^p} \lesssim m^{-\frac{\alpha}{d}} \|f\|_{W^{s,p}}.$$

$\Rightarrow$ Need $m > \varepsilon^{-d}$ to acheive ε approximation error.


D. Yarotsky,
Error bounds for approximations with deep relu networks,
Neural Networks, 94:103–114, 2017.


D. Yarotsky,
Optimal approximation of continuous functions by very deep relu network,
Proceedings of Machine Learning Research, 639–649, 2018.

Neural-Network Approximation

- Is the space of continuous function enough for the neural-network approximation?
- No! The Curse of Dimensionality:

$$\|f_m - f\|_{L^p} \lesssim m^{-\frac{\alpha}{d}} \|f\|_{W^{s,p}}.$$

$\Rightarrow$ Need $m > \varepsilon^{-d}$ to acheive ε approximation error.



D. Yarotsky,
Error bounds for approximations with deep relu networks,
Neural Networks, 94:103–114, 2017.



D. Yarotsky,
Optimal approximation of continuous functions by very deep relu network,
Proceedings of Machine Learning Research, 639–649, 2018.

- We need to identify a proper regularity for a neural-network approximation.

Theorem

- Assume that the target function have the following regularity:

$$\gamma(f) = \int_{\mathbb{R}^d} (1 + |\xi|)|\widehat{f}(\xi)| \, \mathrm{d}\xi < \infty.$$

- Ansatz class: two-layer neural networks

$$\mathcal{N}_n = \left\{ \sum_{k=1}^{n} a_k \sigma(b_k^T x + c_k) \right\}$$

Then for any $n > 0$ there exists a two-layer neural network $f_n \in \mathcal{N}_n$ satisfying

$$\|f_n(x) - f(x)\|_2 \lesssim \frac{\gamma(f)}{\sqrt{n}}.$$

A. Barron,
Universal approximation bounds for superpositions of a sigmoidal function,
IEEE Transactions on Information Theory, 39(3):930–945, 1993.

(Spectral) Barron Space

- For an integrable function f defined on $\Omega \subset \mathbb{R}^d$, we define the quantity

$$\gamma_s(f_e) = \int_{\mathbb{R}^d} (1 + |\xi|)^s |\widehat{f_e}(\xi)| \, \mathrm{d}\xi,$$

where $f_e \in L^1(\mathbb{R}^d)$ is an extension of f to $\mathbb{R}^d$.

(Spectral) Barron Space

- For an integrable function f defined on $\Omega \subset \mathbb{R}^d$, we define the quantity

$$\gamma_s(f_e) = \int_{\mathbb{R}^d} (1 + |\xi|)^s |\widehat{f_e}(\xi)| \, \mathrm{d}\xi,$$

where $f_e \in L^1(\mathbb{R}^d)$ is an extension of f to $\mathbb{R}^d$.

- We define the (spectral) Barron space:

$$\mathcal{B}^s(\Omega) := \{f : \Omega \to \mathbb{R} : \gamma_s(f_e) < \infty \quad \text{for some extension of } f_e \text{ of } f\},$$

which is equipped with the norm

$$\|f\|_{\mathcal{B}^s(\Omega)} := \inf_{f_e|_\Omega = f} \gamma(f_e).$$

where the infimum is taken over all extensions $f_e \in L^1(\mathbb{R}^d)$.

(Spectral) Barron Space

(Spectral) Barron Space

- **(Embedding)** Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Then we have

$$\|f\|_{H^m(\Omega)} \lesssim \|f\|_{\mathcal{B}^m(\Omega)} \lesssim \|f\|_{H^{m+\frac{d}{2}+\varepsilon}(\Omega)}.$$

(Spectral) Barron Space

- **(Embedding)** Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Then we have

$$\|f\|_{H^m(\Omega)} \lesssim \|f\|_{\mathcal{B}^m(\Omega)} \lesssim \|f\|_{H^{m+\frac{d}{2}+\varepsilon}(\Omega)}.$$

- **(Approximability)** Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Then we have
  - **sigmoid, arctan, tanh, SoftPlus** activation function:

$$\inf_{g \in \mathcal{N}_n} \|f - g\|_{H^s(\Omega)} \lesssim \|f\|_{\mathcal{B}^{s+1}} n^{-\frac{1}{2}}.$$

(Spectral) Barron Space

- **(Embedding)** Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Then we have

$$\|f\|_{H^m(\Omega)} \lesssim \|f\|_{\mathcal{B}^m(\Omega)} \lesssim \|f\|_{H^{m+\frac{d}{2}+\varepsilon}(\Omega)}.$$

- **(Approximability)** Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Then we have

  - **sigmoid, arctan, tanh, SoftPlus** activation function:

$$\inf_{g \in \mathcal{N}_n} \|f - g\|_{H^s(\Omega)} \lesssim \|f\|_{\mathcal{B}^{s+1}} n^{-\frac{1}{2}}.$$

  - **ReLU** activation function:

$$\inf_{g \in \mathcal{N}_n} \|f - g\|_{L^\infty(\Omega)} \lesssim \|f\|_{\mathcal{B}^1} n^{-\frac{1}{2}-\frac{1}{d}}.$$

(Spectral) Barron Space

- **(Embedding)** Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Then we have

$$\|f\|_{H^m(\Omega)} \lesssim \|f\|_{\mathcal{B}^m(\Omega)} \lesssim \|f\|_{H^{m+\frac{d}{2}+\varepsilon}(\Omega)}.$$

- **(Approximability)** Let $\Omega \subset \mathbb{R}^d$ be a bounded domain. Then we have

  - **sigmoid, arctan, tanh, SoftPlus** activation function:

$$\inf_{g \in \mathcal{N}_n} \|f - g\|_{H^s(\Omega)} \lesssim \|f\|_{\mathcal{B}^{s+1}} n^{-\frac{1}{2}}.$$

  - **ReLU** activation function:

$$\inf_{g \in \mathcal{N}_n} \|f - g\|_{L^\infty(\Omega)} \lesssim \|f\|_{\mathcal{B}^1} n^{-\frac{1}{2}-\frac{1}{d}}.$$



J. W. Siegel, J. Xu,
Approximation rates for neural networks with general activation functions,
Neural Networks, Volume 128, Pages 313-321, 2020.



J.M. Klusowski, A.R. Barron,
Uniform approximation by neural networks activated by first and second order ridge splines,
arXiv preprint arXiv:1607.07819, 2016.

Generalization

Population Risk: $\mathcal{L}(\theta) := \mathbb{E}_{\boldsymbol{x} \sim \mu^*} \left[\ell(f(\boldsymbol{x}; \theta), y) \right]$

Empirical Risk: $\mathcal{L}_n(\theta) := \dfrac{1}{n} \sum_{i=1}^{n} \ell(f(\boldsymbol{x}_i; \theta), y)$

Generalization

Population Risk: $\mathcal{L}(\theta) := \mathbb{E}_{\boldsymbol{x} \sim \mu^*} \left[\ell(f(\boldsymbol{x}; \theta), y) \right]$

Empirical Risk: $\mathcal{L}_n(\theta) := \dfrac{1}{n} \sum_{i=1}^{n} \ell(f(\boldsymbol{x}_i; \theta), y)$

- We are interested in $\mathcal{L}$, but we can only minimize $\mathcal{L}_n$ in practice. The difference between $\mathcal{L}$ and $\mathcal{L}_n$ (the generalization gap) may be large.

Generalization

Population Risk: $\mathcal{L}(\theta) := \mathbb{E}_{\boldsymbol{x}\sim\mu^*}\left[\ell(f(\boldsymbol{x};\theta), y)\right]$

Empirical Risk: $\mathcal{L}_n(\theta) := \dfrac{1}{n}\sum_{i=1}^{n} \ell(f(\boldsymbol{x}_i;\theta), y)$

- We are interested in $\mathcal{L}$, but we can only minimize $\mathcal{L}_n$ in practice. The difference between $\mathcal{L}$ and $\mathcal{L}_n$ (the generalization gap) may be large.
- We need to control the generalization gap: Rademacher complexity

Generalization

Population Risk: $\mathcal{L}(\theta) := \mathbb{E}_{\boldsymbol{x}\sim\mu^*}\left[\ell(f(\boldsymbol{x};\theta), y)\right]$

Empirical Risk: $\mathcal{L}_n(\theta) := \dfrac{1}{n}\sum_{i=1}^{n} \ell(f(\boldsymbol{x}_i;\theta), y)$

- We are interested in $\mathcal{L}$, but we can only minimize $\mathcal{L}_n$ in practice. The difference between $\mathcal{L}$ and $\mathcal{L}_n$ (the generalization gap) may be large.
- We need to control the generalization gap: Rademacher complexity

Rademacher complexity

Let $\{X_i\}_{i=1}^M$ be i.i.d. random variables. For a function class $\mathcal{F}$, we define the Rademacher complexity by

$$\mathcal{R}_M(\mathcal{F}) = \mathbb{E}_{\{X_i\}_{i=1}^M}\mathbb{E}_{\{\varepsilon_i\}_{i=1}^M}\left[\sup_{f\in\mathcal{F}}\left|\frac{1}{M}\sum_{i=1}^{M}\varepsilon_i f(X_i)\right|\right],$$

where ε_i's denote i.i.d. Bernoulli random variables, which means that $\mathbb{P}(\varepsilon_i = 1) = \mathbb{P}(\varepsilon_i = -1) = \frac{1}{2}$.

Two theorems controlling the generalization gap:

Two theorems controlling the generalization gap:

Theorem

Given a function class $\mathcal{F}$, for any $\delta \in (0,1)$, with probability at lease $1-\delta$ over the i.i.d. random samples $\{\boldsymbol{x}_i\}_{i=1}^M$, we have

$$\sup_{f\in\mathcal{F}} \left| \frac{1}{M}\sum_{i=1}^{M} f(\boldsymbol{x}_i) - \mathbb{E}[f(\boldsymbol{x})] \right| \leq 2\mathcal{R}_M(\mathcal{F}) + 2 \sup_{g,g'\in\mathcal{F}} \|g-g'\|_\infty \sqrt{\frac{2\log(4/\delta)}{M}}.$$

Two theorems controlling the generalization gap:

Theorem

Given a function class $\mathcal{F}$, for any $\delta \in (0,1)$, with probability at lease $1-\delta$ over the i.i.d. random samples $\{\boldsymbol{x}_i\}_{i=1}^M$, we have

$$\sup_{f\in\mathcal{F}} \left| \frac{1}{M}\sum_{i=1}^{M} f(\boldsymbol{x}_i) - \mathbb{E}[f(\boldsymbol{x})] \right| \leq 2\mathcal{R}_M(\mathcal{F}) + 2 \sup_{g,g'\in\mathcal{F}} \|g-g'\|_\infty \sqrt{\frac{2\log(4/\delta)}{M}}.$$

Theorem

Given a function class $\mathcal{F}$ and $\{\boldsymbol{x}_i\}_{i=1}^M$ be a sequence of i.i.d. random samples. Then we have

$$\mathbb{E}\left[\sup_{f\in\mathcal{F}} \left| \frac{1}{M}\sum_{i=1}^{M} f(\boldsymbol{x}_i) - \mathbb{E}[f(\boldsymbol{x})] \right| \right] \leq 2\mathcal{R}_M(\mathcal{F}),$$

where the expectation is taken for the random variables $\{\boldsymbol{x}_i\}_{i=1}^M$.

Generalization

- If the Rademacher complexity is small, it is easier to train the model (efficient learning).

Generalization

- If the Rademacher complexity is small, it is easier to train the model (efficient learning).
- We need to make an upper bound of the Rademacher complexity according to the function class $\mathcal{F}$.

Generalization

- If the Rademacher complexity is small, it is easier to train the model (efficient learning).
- We need to make an upper bound of the Rademacher complexity according to the function class $\mathcal{F}$.
- A natural question is "Is $\mathcal{R}_M(\mathcal{F}) \to 0$ as $M \to \infty$?".

Generalization

- If the Rademacher complexity is small, it is easier to train the model (efficient learning).
- We need to make an upper bound of the Rademacher complexity according to the function class $\mathcal{F}$.
- A natural question is "Is $\mathcal{R}_M(\mathcal{F}) \to 0$ as $M \to \infty$?".
- Roughly speaking, a 'good' function (regularity) can be learned easily (low complexity).

Generalization

- If the Rademacher complexity is small, it is easier to train the model (efficient learning).
- We need to make an upper bound of the Rademacher complexity according to the function class $\mathcal{F}$.
- A natural question is "Is $\mathcal{R}_M(\mathcal{F}) \to 0$ as $M \to \infty$?".
- Roughly speaking, a 'good' function (regularity) can be learned easily (low complexity).

Thoerem

For the function class $\mathcal{F}_{Q,1} = \{g(\cdot;\theta) \in \mathcal{N}_n : \|\theta\|_{\mathcal{P}} \leq Q\}$ (two-layer neural networks) and $\mathcal{F}_{Q,2} = \{g \in \mathcal{W} : \|g\|_{\mathcal{W}} \leq Q\}$ (Barron functions), we have

$$\mathcal{R}_M(\mathcal{F}_{Q,j}) \leq 2Q\sqrt{\frac{2\log(2d+2)}{M}} \quad \text{for } j = 1, 2.$$



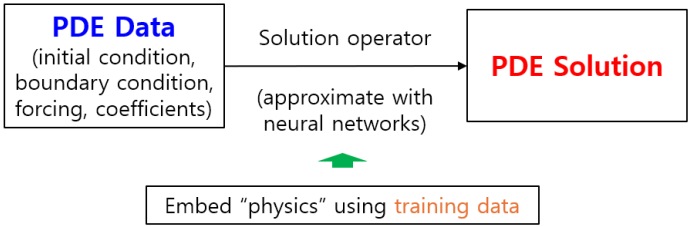
W. E, C. Ma and L. Wu

The Barron Space and the Flow-Induced Function Spaces for Neural Network Models,

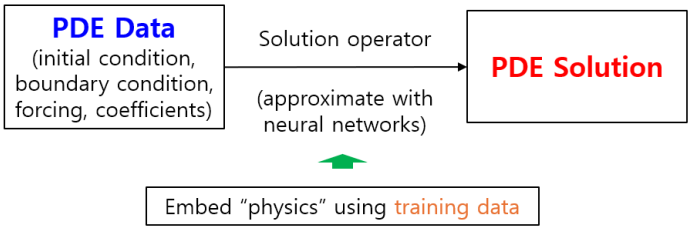
Constructive Approximation, Volume 55, pages 369–406, 2022.

PDE application of Operator Learning

PDE application of Operator Learning

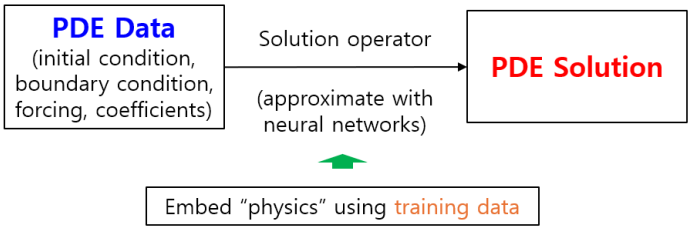


PDE application of Operator Learning



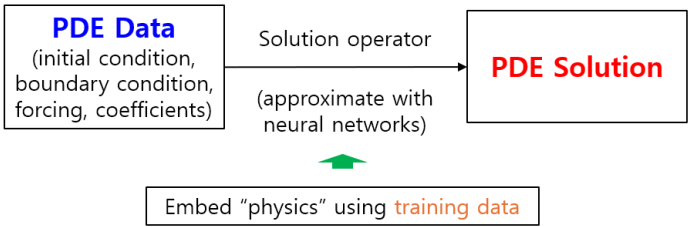
- How to prepare the training data?

PDE application of Operator Learning



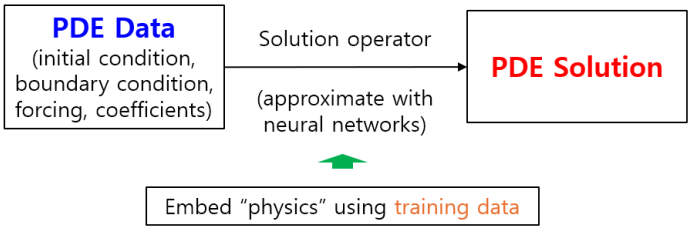
- How to prepare the training data?
  - It is extremely difficult and mostly impossible to compute exact solutions of PDEs.

PDE application of Operator Learning



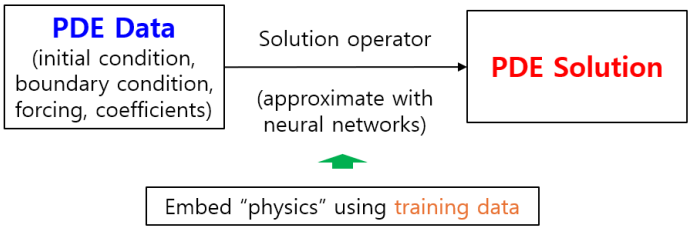
- How to prepare the training data?
  - It is extremely difficult and mostly impossible to compute exact solutions of PDEs.
  - Therefore we use the classical numerical methods (FDM, FEM, FVM) to compute solutions (training data).

PDE application of Operator Learning



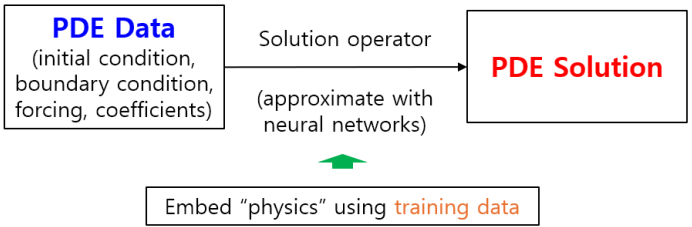
- How to prepare the training data?
  - It is extremely difficult and mostly impossible to compute exact solutions of PDEs.
  - Therefore we use the classical numerical methods (FDM, FEM, FVM) to compute solutions (training data).
  - But we need to use very fine meshes: computationally very expensive.

PDE application of Operator Learning



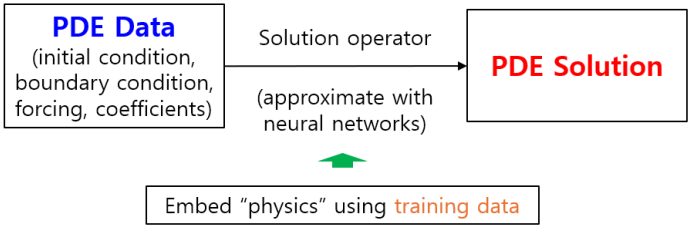
- How to prepare the training data?
  - It is extremely difficult and mostly impossible to compute exact solutions of PDEs.
  - Therefore we use the classical numerical methods (FDM, FEM, FVM) to compute solutions (training data).
  - But we need to use very fine meshes: computationally very expensive.
  - Once the training is finished, the operator learning method can make a real-time prediction, but reaching that stage is not easy.

PDE application of Operator Learning



- How to prepare the training data?
  - It is extremely difficult and mostly impossible to compute exact solutions of PDEs.
  - Therefore we use the classical numerical methods (FDM, FEM, FVM) to compute solutions (training data).
  - But we need to use very fine meshes: computationally very expensive.
  - Once the training is finished, the operator learning method can make a real-time prediction, but reaching that stage is not easy.
  - Adequacy issue: Do we really need "supervised operator learning model"?

PDE application of Operator Learning



- How to prepare the training data?
  - It is extremely difficult and mostly impossible to compute exact solutions of PDEs.
  - Therefore we use the classical numerical methods (FDM, FEM, FVM) to compute solutions (training data).
  - But we need to use very fine meshes: computationally very expensive.
  - Once the training is finished, the operator learning method can make a real-time prediction, but reaching that stage is not easy.
  - Adequacy issue: Do we really need "supervised operator learning model"?
- Next step: Can we learn an operator in an unsupervised manner?

Model Problem

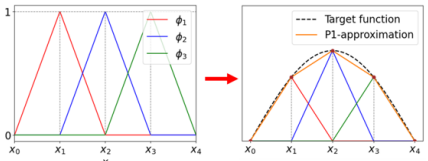
$$
\begin{aligned}
-\Delta u(x) + u(x) &= f(x) && \text{in } D, \\
u(x) &= 0 && \text{on } \partial D.
\end{aligned}
$$

Model Problem

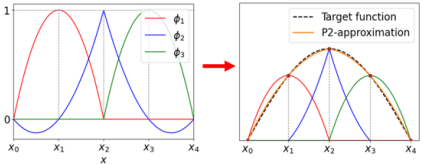
$$
\begin{aligned}
-\Delta u(x) + u(x) &= f(x) \qquad \text{in } D, \\
u(x) &= 0 \qquad \text{on } \partial D.
\end{aligned}
$$

- **Galerkin approximation**: Find $u_h \in \mathbb{V}_h := \operatorname{span}\{\phi_1, \cdots, \phi_{N_h}\}$ such that

$$
B[u_h, v_h] := (\nabla u_h, \nabla v_h) + (u_h, v_h) = (f, v_h) =: \ell(v_h) \qquad \forall v_h \in \mathbb{V}_h.
$$



(a) P1 basis and P1 approximation of $y = \sin x$ (Rel. L^2-error 0.056)



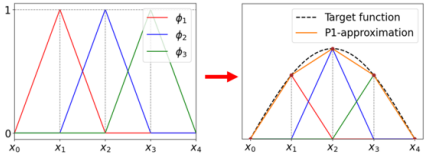
(b) P2 basis and P2 approximation of $y = \sin x$ (Rel. L^2-error 0.021)

Model Problem

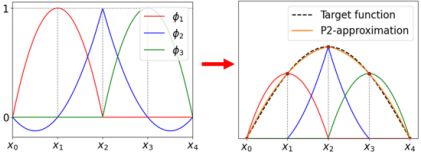
$$
\begin{aligned}
-\Delta u(x) + u(x) &= f(x) \qquad \text{in } D, \\
u(x) &= 0 \qquad \text{on } \partial D.
\end{aligned}
$$

- **Galerkin approximation**: Find $u_h \in \mathbb{V}_h := \mathrm{span}\{\phi_1, \cdots, \phi_{N_h}\}$ such that

$$B[u_h, v_h] := (\nabla u_h, \nabla v_h) + (u_h, v_h) = (f, v_h) =: \ell(v_h) \qquad \forall v_h \in \mathbb{V}_h.$$



(a) P1 basis and P1 approximation of $y = \sin x$ (Rel. L^2-error 0.056)



(b) P2 basis and P2 approximation of $y = \sin x$ (Rel. L^2-error 0.021)

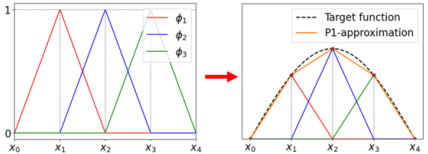
- $u_h(x) = \sum_{i=1}^{N_h} \alpha_i \phi_i(x) \Rightarrow$ Solve the linear algebraic equation $A\alpha = F$, where $A_{ij} = B[\phi_i, \phi_j]$, $\ell(\phi_j) = F_j$.

Model Problem

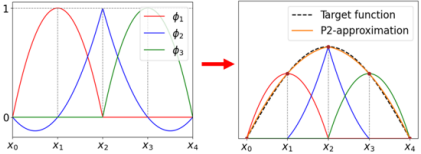
$$
\begin{aligned}
-\Delta u(x) + u(x) &= f(x) \qquad \text{in } D, \\
u(x) &= 0 \qquad \text{on } \partial D.
\end{aligned}
$$

- **Galerkin approximation**: Find $u_h \in \mathbb{V}_h := \mathrm{span}\{\phi_1, \cdots, \phi_{N_h}\}$ such that

$$
B[u_h, v_h] := (\nabla u_h, \nabla v_h) + (u_h, v_h) = (f, v_h) =: \ell(v_h) \qquad \forall v_h \in \mathbb{V}_h.
$$



(a) P1 basis and P1 approximation of $y = \sin x$ (Rel. L^2-error 0.056)



(b) P2 basis and P2 approximation of $y = \sin x$ (Rel. L^2-error 0.021)

- $u_h(x) = \sum_{i=1}^{N_h} \alpha_i \phi_i(x) \Rightarrow$ Solve the linear algebraic equation $A\alpha = F$, where $A_{ij} = B[\phi_i, \phi_j]$, $\ell(\phi_j) = F_j$.

  (Some issues of the condition number $\kappa(A)$ when inverting A)

Finite Element Operator Network (FEONet)

- **FEONet solution**: $u_h(x) = \sum_{i=1}^{N_h} \alpha_i(\omega)\phi_i(x)$ (α: neural network, ϕ_i: basis function)

Finite Element Operator Network (FEONet)

- **FEONet solution**: $u_h(x) = \sum_{i=1}^{N_h} \alpha_i(\omega)\phi_i(x)$ (α: neural network, ϕ_i: basis function)
- **Unsupervised Legendre–Galerkin neural networks (ULGNet)** : Legendre polynomials.



J. Choi, N. Kim and Y. Hong,
Unsupervised Legendre-Galerkin Neural Network for Solving Partial Differential Equations,
IEEE Access, vol. 11, pp. 23433-23446, 2023.



S. Ko, S. Yun and Y. Hong,
Convergence analysis of unsupervised Legendre-Galerkin neural networks for linear second-order elliptic PDEs,
submitted, 2023.



J. Choi, T. Yun, N. Kim, Y. Hong
Spectral operator learning for parametric PDEs without data reliance,
Computer Methods in Applied Mechanics and Engineering Volume 420, 2024.

Finite Element Operator Network (FEONet)

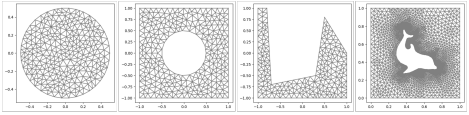
- **FEONet solution**: $u_h(x) = \sum_{i=1}^{N_h} \alpha_i(\omega)\phi_i(x)$ (α: neural network, ϕ_i: basis function)
- **Unsupervised Legendre–Galerkin neural networks (ULGNet)** : Legendre polynomials.

J. Choi, N. Kim and Y. Hong,
Unsupervised Legendre-Galerkin Neural Network for Solving Partial Differential Equations,
IEEE Access, vol. 11, pp. 23433-23446, 2023.

S. Ko, S. Yun and Y. Hong,
Convergence analysis of unsupervised Legendre-Galerkin neural networks for linear second-order elliptic PDEs,
submitted, 2023.

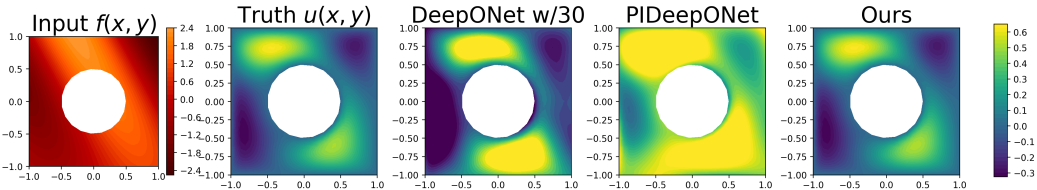
J. Choi, T. Yun, N. Kim, Y. Hong
Spectral operator learning for parametric PDEs without data reliance,
Computer Methods in Applied Mechanics and Engineering Volume 420, 2024.

- **Finite Element Operator Network (FEONet)** : Finite element nodal basis.



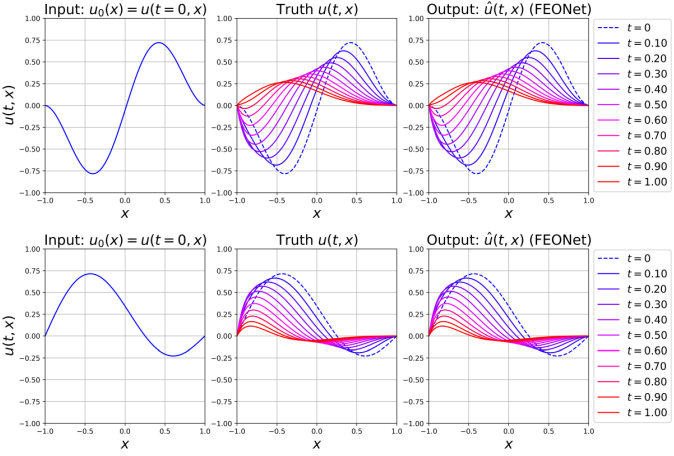
(a) Circle (b) Square with a hole (c) Polygon (d) Dolphin

Numerical Experiments

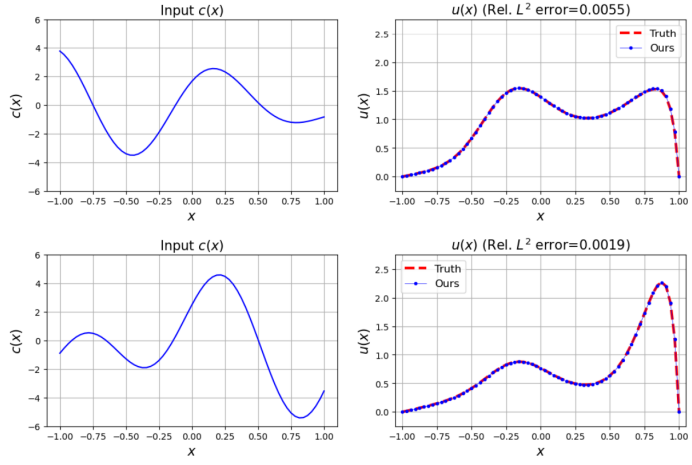


Model (#Train data)	Domain I	Domain II	Domain III	BC I	BC II	Eq I	Eq II
DON (supervised, w/30)	27.15±1.16	51.21±3.58	53.92±4.59	21.75±1.19	22.75±1.05	24.38±1.37	10.26±0.14
DON (supervised, w/300)	2.10±0.75	5.62±0.37	6.22±0.96	0.68±0.11	0.96±0.06	0.76±0.10	**0.20**±0.09
DON (supervised, w/3000)	**0.69**±0.17	4.75±0.75	6.20±1.00	0.53±0.36	0.33±0.09	0.33±0.27	0.24±0.13
PIDeepONet (w/o labeled data)	9.80±9.41	101.03±167.46	32.89±6.34	1.51±0.46	1.43±0.45	19.41±11.30	2.66±0.71
Ours (w/o labeled data)	1.24±0.00	**1.76**±0.03	**0.51**±0.00	**0.13**±0.01	**0.32**±0.03	**0.13**±0.01	0.54±0.07

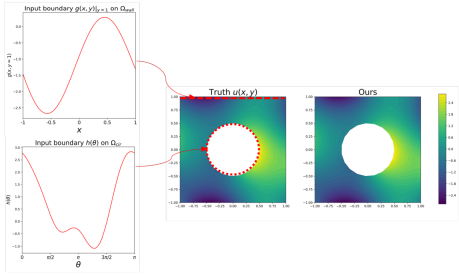
Numerical Experiments (initial data input)

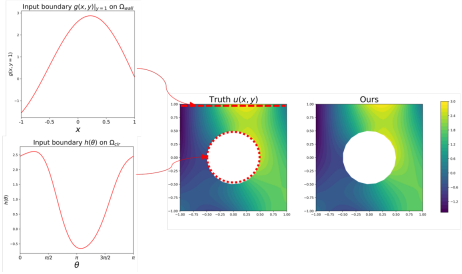


Numerical Experiments (variable coefficient input)



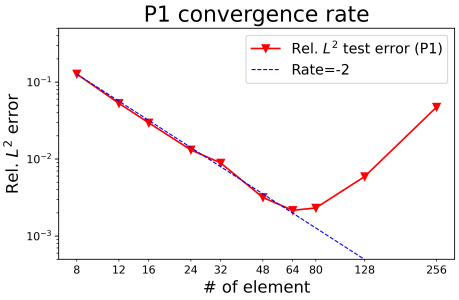
Numerical Experiments (boundary data input)

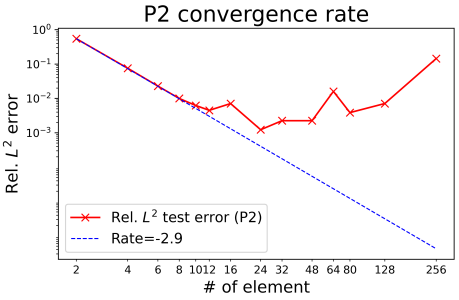




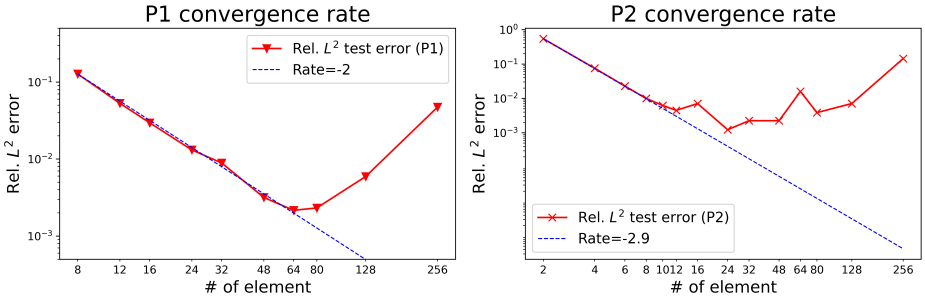
But something mysterious happens...

But something mysterious happens...



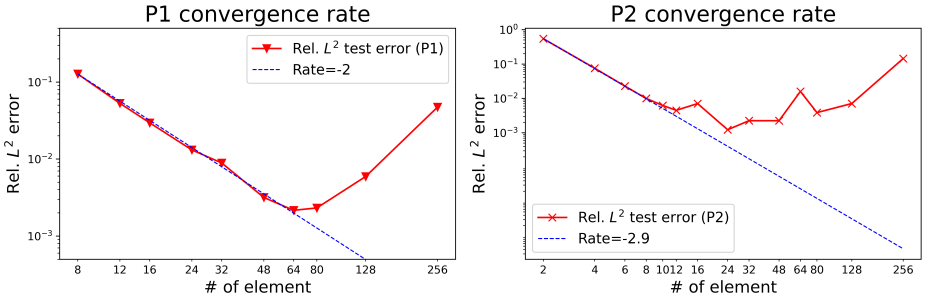


But something mysterious happens...



- Why does the phenomenon of errors increasing again at a certain point occur?

But something mysterious happens...



- Why does the phenomenon of errors increasing again at a certain point occur?
- The objective is to investigate the underlying principle of FEONet based on the mathematical analysis.

Total error: $u - u_{h,n,M} = \underbrace{(u - u_h)}_{\text{FEM error}} + \underbrace{(u_h - u_{h,n})}_{\text{approximation error}} + \underbrace{(u_{h,n} - u_{h,n,M})}_{\text{generalization error}}$

Total error: $u - u_{h,n,M} = \underbrace{(u - u_h)}_{\text{FEM error}} + \underbrace{(u_h - u_{h,n})}_{\text{approximation error}} + \underbrace{(u_{h,n} - u_{h,n,M})}_{\text{generalization error}}$

- **FEM error**: The error arises from the finite element approximation whos mathematical analysis is fairly well-known.

Total error: $u - u_{h,n,M} = \underbrace{(u - u_h)}_{\text{FEM error}} + \underbrace{(u_h - u_{h,n})}_{\text{approximation error}} + \underbrace{(u_{h,n} - u_{h,n,M})}_{\text{generalization error}}$

- **FEM error**: The error arises from the finite element approximation whos mathematical analysis is fairly well-known.

- **Approximation error**: This error occurs when the true coefficient α^* is approximated by a deep neural network.

Total error: $u - u_{h,n,M} = \underbrace{(u - u_h)}_{\text{FEM error}} + \underbrace{(u_h - u_{h,n})}_{\text{approximation error}} + \underbrace{(u_{h,n} - u_{h,n,M})}_{\text{generalization error}}$

- **FEM error**: The error arises from the finite element approximation whos mathematical analysis is fairly well-known.
- **Approximation error**: This error occurs when the true coefficient α^* is approximated by a deep neural network.
- **Generalization error**: This error measures how well the predicted solution $u_{h,n,M}$ trained with M random samples generalizes for other samples that were not used in the training process.

Theoretical Analysis of FEONet

Approximation error (Céa's lemma for the neural network approximation)

Let $\kappa(A)$ be a condition number of the given finite element matrix A. Then we have

$$\|\alpha^* - \widehat{\alpha}_n^{\mathcal{L}}\|_{L^1(\Omega)} \leq \kappa(A) \inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_{L^1(\Omega)}.$$

Theoretical Analysis of FEONet

Approximation error (Céa's lemma for the neural network approximation)

Let $\kappa(A)$ be a condition number of the given finite element matrix A. Then we have

$$\|\alpha^* - \widehat{\alpha}_n^{\mathcal{L}}\|_{L^1(\Omega)} \leq \kappa(A) \inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_{L^1(\Omega)}.$$

Generalization error

Let $\kappa(A)$ be a condition number of the given finite element matrix A. Then we have

$$\mathbb{E}\left[\|\widehat{\alpha}_n^{\mathcal{L}} - \widehat{\alpha}_{n,M}^{\mathcal{L}}\|_{L^1(\Omega)}\right] \lesssim \kappa(A)^{d/2} \mathcal{R}_M(\mathcal{F}_n^{\mathcal{L}}) + \kappa(A) \inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_{L^1(\Omega)}.$$

Theoretical Analysis of FEONet

Approximation error (Céa's lemma for the neural network approximation)

Let $\kappa(A)$ be a condition number of the given finite element matrix A. Then we have

$$\|\alpha^* - \widehat{\alpha}_n^{\mathcal{L}}\|_{L^1(\Omega)} \leq \kappa(A) \inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_{L^1(\Omega)}.$$

Generalization error

Let $\kappa(A)$ be a condition number of the given finite element matrix A. Then we have

$$\mathbb{E}\left[\|\widehat{\alpha}_n^{\mathcal{L}} - \widehat{\alpha}_{n,M}^{\mathcal{L}}\|_{L^1(\Omega)}\right] \lesssim \kappa(A)^{d/2} \mathcal{R}_M(\mathcal{F}_n^{\mathcal{L}}) + \kappa(A) \inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_{L^1(\Omega)}.$$

Error estimate for the FEONet using Pℓ-element:

$$\mathbb{E}\left[\|u - u_{h,n,M}\|_{L^1(\Omega;L^2(D))}\right] \lesssim h^{\ell+1} + \kappa(A) \inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_{L^1(\Omega)} + \kappa(A)^{d/2} \mathcal{R}_M(\mathcal{F}_n^{\mathcal{L}})$$

Theoretical Analysis of FEONet

Approximation error (Céa's lemma for the neural network approximation)

Let $\kappa(A)$ be a condition number of the given finite element matrix A. Then we have

$$\|\alpha^* - \widehat{\alpha}_n^{\mathcal{L}}\|_{L^1(\Omega)} \leq \kappa(A) \inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_{L^1(\Omega)}.$$

Generalization error

Let $\kappa(A)$ be a condition number of the given finite element matrix A. Then we have

$$\mathbb{E}\left[\|\widehat{\alpha}_n^{\mathcal{L}} - \widehat{\alpha}_{n,M}^{\mathcal{L}}\|_{L^1(\Omega)}\right] \lesssim \kappa(A)^{d/2} \mathcal{R}_M(\mathcal{F}_n^{\mathcal{L}}) + \kappa(A) \inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_{L^1(\Omega)}.$$

Error estimate for the FEONet using Pℓ-element:

$$\mathbb{E}\left[\|u - u_{h,n,M}\|_{L^1(\Omega;L^2(D))}\right] \lesssim h^{\ell+1} + \kappa(A) \inf_{\alpha \in \mathcal{N}_n} \|\alpha - \alpha^*\|_{L^1(\Omega)} + \kappa(A)^{d/2} \mathcal{R}_M(\mathcal{F}_n^{\mathcal{L}})$$

Barron theory needed!

Theorem (regularity of coefficients)

Assume that $f(x,\omega) \in L^1(D; H^s(\Omega))$ with $s > m/2 + 2$. Then the target coefficient (finite element coefficient) α_j^* is contained in $\mathcal{B}^2(\Omega)$ with the estimate

$$\|\alpha_j^*\|_{\mathcal{B}^2(\Omega)} \lesssim \kappa(A)^{d/2}.$$

Theoretical Analysis of FEONet

Theorem (Error estimate for the FEONet)

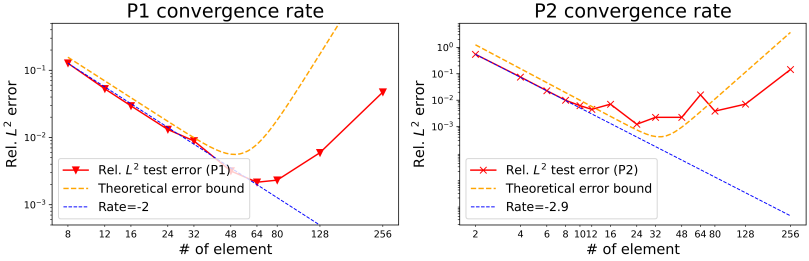
If we use the $(P\ell)$-element, the predicted solution $u_{h,n,M}$ by the FEONet satisfies the following error estimate:

$$\mathbb{E}\left[\|u - u_{h,n,M}\|_{L^1(\Omega;L^2(D))}\right] \lesssim h^{\ell+1} + \frac{\kappa(A)^{1+d}}{\sqrt{n}} + \frac{\kappa(A)^{1+3d/2}}{\sqrt{M}}.$$

Theorem (Error estimate for the FEONet)

If we use the $(P\ell)$-element, the predicted solution $u_{h,n,M}$ by the FEONet satisfies the following error estimate:

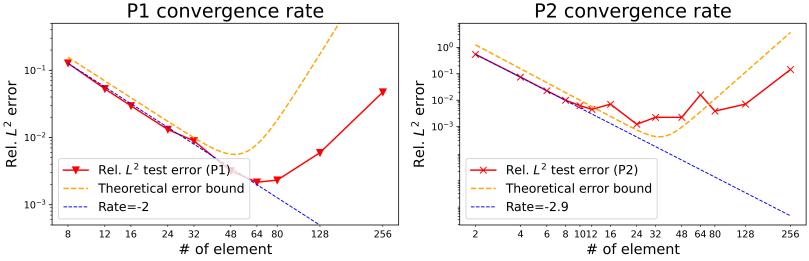
$$\mathbb{E}\left[\|u - u_{h,n,M}\|_{L^1(\Omega;L^2(D))}\right] \lesssim h^{\ell+1} + \frac{\kappa(A)^{1+d}}{\sqrt{n}} + \frac{\kappa(A)^{1+3d/2}}{\sqrt{M}}.$$



Theorem (Error estimate for the FEONet)

If we use the $(P\ell)$-element, the predicted solution $u_{h,n,M}$ by the FEONet satisfies the following error estimate:

$$\mathbb{E}\left[\|u - u_{h,n,M}\|_{L^1(\Omega;L^2(D))}\right] \lesssim h^{\ell+1} + \frac{\kappa(A)^{1+d}}{\sqrt{n}} + \frac{\kappa(A)^{1+3d/2}}{\sqrt{M}}.$$

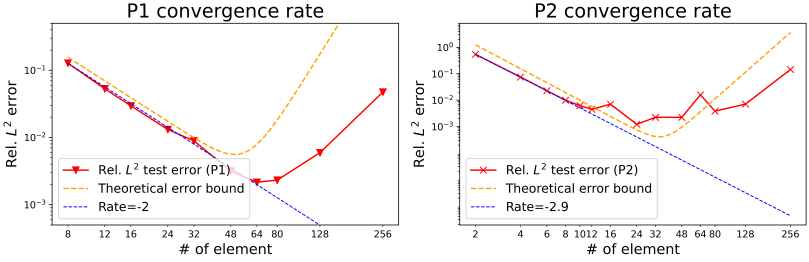


- **(Theory-guided strategy 1)** Take $n \to \infty$ and $M \to \infty$.

Theorem (Error estimate for the FEONet)

If we use the $(P\ell)$-element, the predicted solution $u_{h,n,M}$ by the FEONet satisfies the following error estimate:

$$\mathbb{E}\left[\|u - u_{h,n,M}\|_{L^1(\Omega;L^2(D))}\right] \lesssim h^{\ell+1} + \frac{\kappa(A)^{1+d}}{\sqrt{n}} + \frac{\kappa(A)^{1+3d/2}}{\sqrt{M}}.$$



- **(Theory-guided strategy 1)** Take $n \to \infty$ and $M \to \infty$.
- **(Theory-guided strategy 2)** Use a preconditioned loss: replace $|A\alpha - F|$ by $|P^{-1}A\alpha - P^{-1}F|$

Numerical Experiments

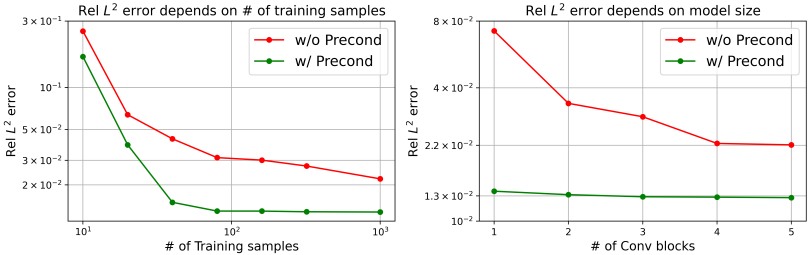


Figure: The relative L^2 errors resulting from varying the number of training samples and the model size.

Numerical Experiments

# of elements		8	12	16	24	32	48	64	80	128	256
CD Eq.	w/o precond	0.12731	0.05301	0.02958	0.01316	0.00890	**0.00318**	0.00216	0.00232	0.00593	0.04736
	w precond	**0.12728**	**0.05291**	**0.02887**	**0.01231**	**0.00780**	0.00584	**0.00170**	**0.00110**	**0.00048**	**0.00021**

Table: Numerical errors against the number of P1 elements for convection-diffusion equation.

# of elements		2	4	6	8	10	12	16	24	32	48	64	80	128	256
CD Eq.	w/o precond	**0.54603**	**0.07546**	0.02294	0.01000	**0.00624**	0.00451	0.00709	**0.00123**	0.00225	0.00225	0.01603	0.00387	0.00711	0.14480
	w precond	0.54614	0.07576	**0.02253**	**0.00949**	0.00738	**0.00204**	**0.00479**	0.00573	**0.00034**	**0.00034**	**0.00026**	**0.00039**	**0.00027**	**0.00239**

Table: Numerical errors against the number of P2 elements for convection-diffusion equation.

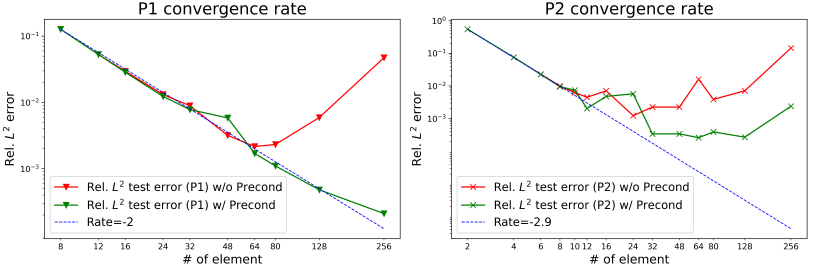
Numerical Experiments

# of elements		8	12	16	24	32	48	64	80	128	256
CD Eq.	w/o precond	0.12731	0.05301	0.02958	0.01316	0.00890	**0.00318**	0.00216	0.00232	0.00593	0.04736
	w precond	**0.12728**	**0.05291**	**0.02887**	**0.01231**	**0.00780**	0.00584	**0.00170**	**0.00110**	**0.00048**	**0.00021**

Table: Numerical errors against the number of P1 elements for convection-diffusion equation.

# of elements		2	4	6	8	10	12	16	24	32	48	64	80	128	256
CD Eq.	w/o precond	**0.54603**	**0.07546**	0.02294	0.01000	**0.00624**	0.00451	0.00709	**0.00123**	0.00225	0.00225	0.01603	0.00387	0.00711	0.14480
	w precond	0.54614	0.07576	**0.02253**	**0.00949**	0.00738	**0.00204**	**0.00479**	0.00573	**0.00034**	**0.00034**	**0.00026**	**0.00039**	**0.00027**	**0.00239**

Table: Numerical errors against the number of P2 elements for convection-diffusion equation.



Numerical Experiments

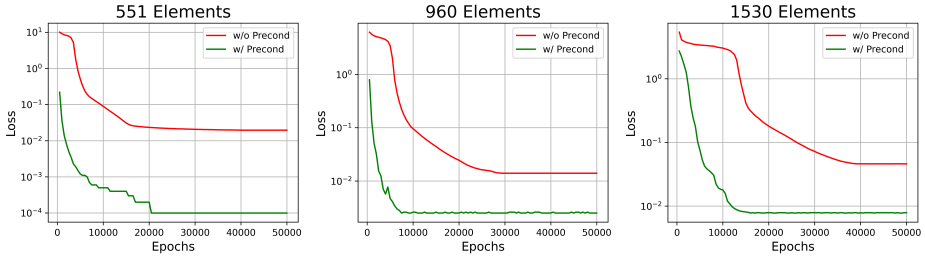


Figure: The FEONet prediction for 2D Poisson equation without or with preconditioning.

Summary

- **How to improve the performance?**

Summary

- **How to improve the performance?**
  - Machine Learning Model $\longrightarrow \underbrace{\text{(Black Box)} \longrightarrow \text{(Repetitive Trial)}}_{\text{empirical process}}$

Summary

- **How to improve the performance?**
  - Machine Learning Model $\longrightarrow \underbrace{\text{(Black Box)} \longrightarrow \text{(Repetitive Trial)}}_{\text{empirical process}}$
  - Machine Learning Model $\longrightarrow \underbrace{\text{(Theoretical Analysis)} \longrightarrow \text{(Insight from Theory)}}_{\text{theory-guided strategy}}$

Summary

- **How to improve the performance?**
  - Machine Learning Model $\longrightarrow \underbrace{\text{(Black Box)} \longrightarrow \text{(Repetitive Trial)}}_{\text{empirical process}}$
  - Machine Learning Model $\longrightarrow \underbrace{\text{(Theoretical Analysis)} \longrightarrow \text{(Insight from Theory)}}_{\text{theory-guided strategy}}$

- **Understanding the mechanism behind the empirical success is important.**

Summary

- **How to improve the performance?**
  - Machine Learning Model $\longrightarrow \underbrace{\text{(Black Box)} \longrightarrow \text{(Repetitive Trial)}}_{\text{empirical process}}$
  - Machine Learning Model $\longrightarrow \underbrace{\text{(Theoretical Analysis)} \longrightarrow \text{(Insight from Theory)}}_{\text{theory-guided strategy}}$
- **Understanding the mechanism behind the empirical success is important.**
- **Developing new mathematical frameworks and analytical tools is needed.**

Summary

- **How to improve the performance?**
  - Machine Learning Model $\longrightarrow \underbrace{\text{(Black Box)} \longrightarrow \text{(Repetitive Trial)}}_{\text{empirical process}}$
  - Machine Learning Model $\longrightarrow \underbrace{\text{(Theoretical Analysis)} \longrightarrow \text{(Insight from Theory)}}_{\text{theory-guided strategy}}$
- **Understanding the mechanism behind the empirical success is important.**
- **Developing new mathematical frameworks and analytical tools is needed.**

Mathematics is important in machine learning!

Thank you very much.