Solving the Biharmonic Equation by Deep Neural Network

Jia-Wei Liao, Yu-Hsi Chen, Woan-Rong Huang
Advisor: Ming-Chih Lai

Department of Applied Mathematics
National Yang Ming Chiao Tung University

February 4, 2024

Outline

Problem

Poisson equation:

$$\begin{cases} \Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \partial\Omega \end{cases}$$

Problem

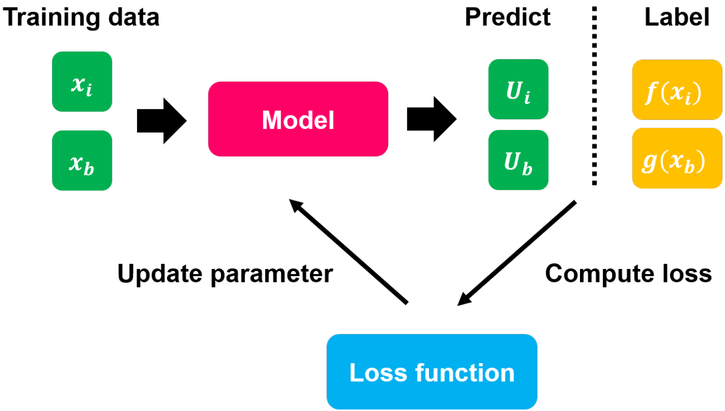
Poisson equation:

$$\begin{cases} \Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \partial\Omega \end{cases}$$

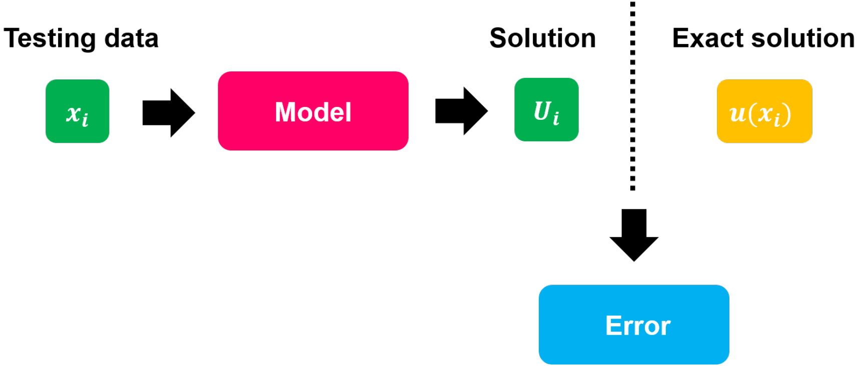
Biharmonic equation:

$$\begin{cases} \Delta^2 u = f, & \text{in } \Omega, \\ u = g_0, & \text{on } \partial\Omega, \\ \dfrac{\partial u}{\partial n} = g_1, & \text{on } \partial\Omega \end{cases} \implies \begin{cases} \Delta u = p, & \text{in } \Omega, \\ \Delta p = f, & \text{in } \Omega, \\ u = g_0, & \text{on } \partial\Omega, \\ \dfrac{\partial u}{\partial n} = g_1, & \text{on } \partial\Omega \end{cases}$$

Training process



Testing process



Neural network (NN)

Question: Why functions can be approximated by neural network? [1]

Theorem (Universal Approximation Theorem With ReLU Network)

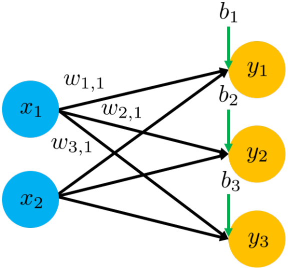
For any Lebesgue-integrable function $f : \mathbb{R}^n \to \mathbb{R}$ and any $\varepsilon > 0$, there exists a fully-connected ReLU network $\mathcal{Q}$ with width $\leq n+4$ and depth $\leq 4n+1$ such that the function $F_{\mathcal{Q}}$ represented by this network satisfies

$$\int_{\mathbb{R}^n} |f(x) - F_{\mathcal{Q}}| dx < \varepsilon$$

[1] Lu et al., The Expressive power of Neural Networks: A View from the Width, NIPS 2017

Fully connected layer

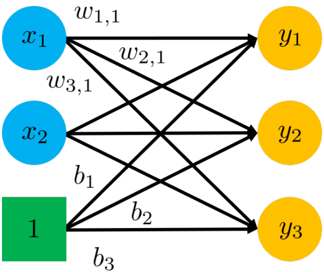
Version 1:



$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} \\ w_{2,1} & w_{2,2} \\ w_{3,1} & w_{3,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

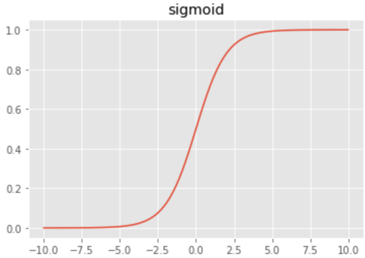
Fully connected layer

Version 2:

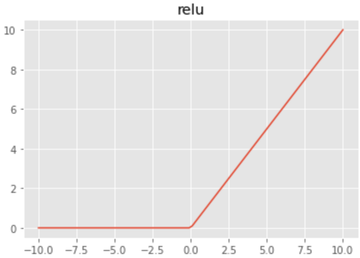


$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} w_{1,1} & w_{1,2} & b_1 \\ w_{2,1} & w_{2,2} & b_2 \\ w_{3,1} & w_{3,2} & b_3 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix}$$

Activation function

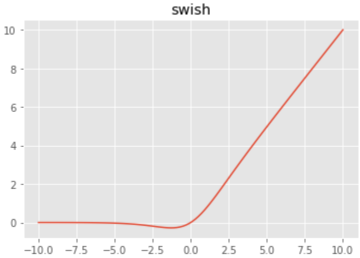


$$\text{Sigmoid}(x) = \frac{1}{1+e^{-x}}$$

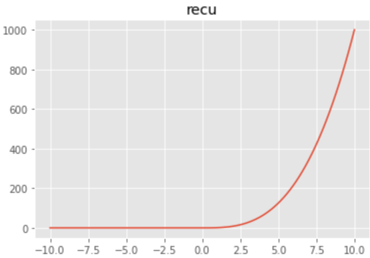


$$\text{ReLU}(x) = \max(x, 0)$$

Activation function

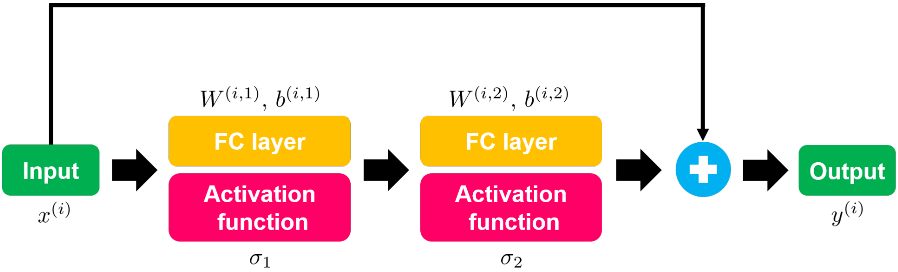


$$\text{Swish}(x) = \frac{x}{1+e^{-x}}$$



$$\text{ReCU}(x) = \max(x^3, 0)$$

Residual Network



$$y^{(i)} = \sigma_2\left(W^{(i,2)} \cdot \sigma_1(W^{(i,1)}x + b^{(i,1)}) + b^{(i,2)}\right) + x^{(i)}$$

Optimization

Gradient decent:

$$\theta_t = \theta_{t-1} - \gamma \nabla_\theta \mathcal{L}(u;\theta)$$

where

- $\mathcal{L}$: loss function.
- θ : parameters in the Neural Network.
- γ : learning rate.

Optimization

Adam algorithm (ICLR 2015)

Let $\mathcal{L}(\theta)$ be the objective function with parameters θ, β_1, β_2 be the exponential decay rates for the moment estimates, γ be the learning rate and $\varepsilon = 10^{-8}$.

1. $m_t = \beta_1 m_{t-1} + (1-\beta_1)\nabla_\theta \mathcal{L}(\theta_{t-1})$
2. $v_t = \beta_2 v_{t-1} + (1-\beta_2)(\nabla_\theta \mathcal{L}(\theta_{t-1}))^2$
3. $\hat{m}_t = \dfrac{m_{t-1}}{1-\beta_1^t}$
4. $\hat{v}_t = \dfrac{v_{t-1}}{1-\beta_2^t}$
5. $\theta_t = \theta_{t-1} - \gamma \dfrac{\hat{m}_t}{\sqrt{\hat{v}_t} + \varepsilon}$

Forward propagation and backward propagation

Motivation:

- Minimize the loss function by using gradient descent.

Forward propagation and backward propagation

Motivation:

- Minimize the loss function by using gradient descent.

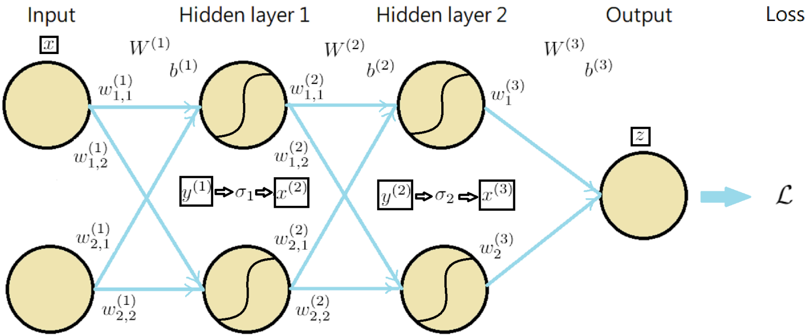
Approach:

- Build a small neural network as defined in the architecture below.
- Use forward propagation to get predicted value and calculate the loss.
- Use backward propagation and adjust weights and bias accordingly.
- Repeat forward and backward steps until the stop criterion is satisfied.

Architecture:

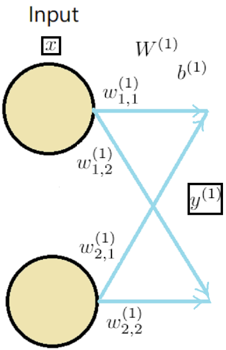
- Build a Feed Forward neural network with 2 hidden layers.
  All layers have 2 Neurons.

Forward propagation



Forward propagation

Matrix operation $W^{(1)}$ and $b^{(1)}$:

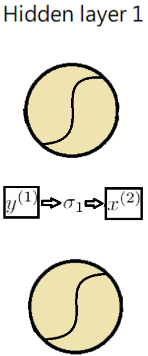


$$y^{(1)} = W^{(1)}x + b^{(1)}$$

$$\begin{bmatrix} y^{(1)}_1 \\ y^{(1)}_2 \end{bmatrix} = \begin{bmatrix} w^{(1)}_{1,1} & w^{(1)}_{1,2} \\ w^{(1)}_{2,1} & w^{(1)}_{2,2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} b^{(2)}_1 \\ b^{(2)}_2 \end{bmatrix}$$

Forward propagation

Activation function σ_1:

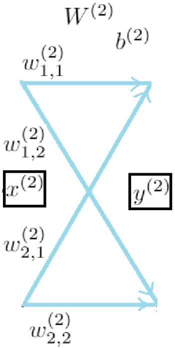


$$x^{(2)} = \sigma_1(y^{(1)})$$

$$\begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} = \begin{bmatrix} \sigma_1(y_1^{(1)}) \\ \sigma_1(y_2^{(1)}) \end{bmatrix}$$

Forward propagation

Matrix operation $W^{(2)}$ and $b^{(2)}$:

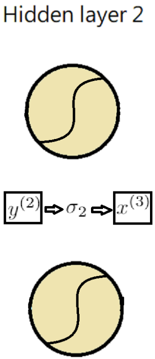


$$y^{(2)} = W^{(2)} x^{(2)} + b^{(2)}$$

$$\begin{bmatrix} y_1^{(2)} \\ y_2^{(2)} \end{bmatrix} = \begin{bmatrix} w_{1,1}^{(2)} & w_{1,2}^{(2)} \\ w_{2,1}^{(2)} & w_{2,2}^{(2)} \end{bmatrix} \begin{bmatrix} x_1^{(2)} \\ x_2^{(2)} \end{bmatrix} + \begin{bmatrix} b_1^{(1)} \\ b_2^{(1)} \end{bmatrix}$$

Forward propagation

Activation function σ_2:

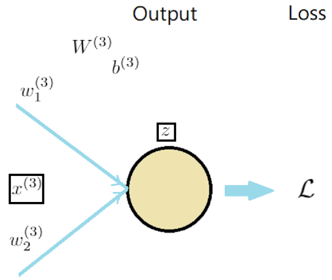


$$x^{(3)} = \sigma_2(y^{(2)})$$

$$\begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} = \begin{bmatrix} \sigma_2(y_1^{(2)}) \\ \sigma_2(y_2^{(2)}) \end{bmatrix}$$

Forward propagation

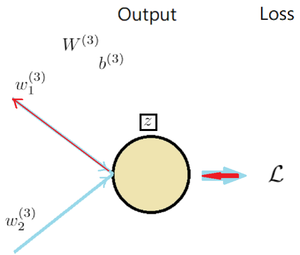
Matrix operation $W^{(3)}$ and $b^{(3)}$:



$$z = W^{(3)} x^{(3)} + b^{(3)}$$

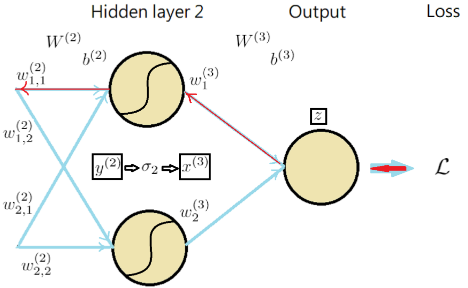
$$z = \begin{bmatrix} w_1^{(3)} & w_2^{(3)} \end{bmatrix} \begin{bmatrix} x_1^{(3)} \\ x_2^{(3)} \end{bmatrix} + b^{(3)}$$

Backward propagation



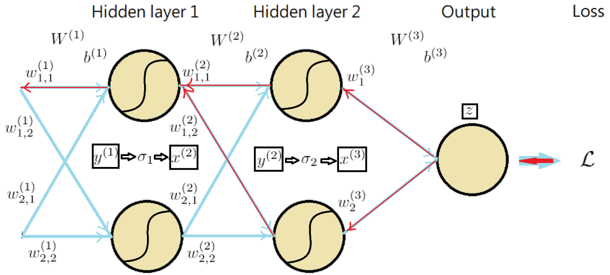
$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial w_i^{(3)}} = \dfrac{\partial \mathcal{L}}{\partial z} \cdot \dfrac{\partial z}{\partial w_i^{(3)}} = \dfrac{\partial \mathcal{L}}{\partial z} \cdot x_i^{(3)} \\ \dfrac{\partial \mathcal{L}}{\partial b^{(3)}} = \dfrac{\partial \mathcal{L}}{\partial z} \cdot \dfrac{\partial z}{\partial b^{(3)}} = \dfrac{\partial \mathcal{L}}{\partial z} \cdot 1 \end{cases}$$

Backward propagation



$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial w_{i,j}^{(2)}} = \dfrac{\partial \mathcal{L}}{\partial z} \cdot \dfrac{\partial z}{\partial x_j^{(3)}} \cdot \dfrac{\partial x_j^{(3)}}{\partial y_j^{(2)}} \cdot \dfrac{\partial y_j^{(2)}}{\partial w_{i,j}^{(2)}} = \dfrac{\partial \mathcal{L}}{\partial z} \cdot w_j^{(3)} \cdot \sigma_2'(y_j^{(2)}) \cdot x_i^{(2)} \\ \dfrac{\partial \mathcal{L}}{\partial b_i^{(2)}} = \dfrac{\partial \mathcal{L}}{\partial z} \cdot \dfrac{\partial z}{\partial x_i^{(3)}} \cdot \dfrac{\partial x_i^{(3)}}{\partial y_i^{(2)}} \cdot \dfrac{\partial y_i^{(2)}}{\partial b_i^{(2)}} \cdot = \dfrac{\partial \mathcal{L}}{\partial z} \cdot w_i^{(3)} \cdot \sigma_2'(y_i^{(2)}) \cdot 1 \end{cases}$$

Backward propagation



$$\begin{cases} \dfrac{\partial \mathcal{L}}{\partial w_{i,j}^{(1)}} = \dfrac{\partial \mathcal{L}}{\partial z} \cdot \dfrac{\partial z}{\partial x_*^{(3)}} \cdot \dfrac{\partial x_*^{(3)}}{\partial y^{(2)}} \cdot \dfrac{\partial y^{(2)}}{\partial x^{(2)}} \cdot \dfrac{\partial x^{(2)}}{\partial y^{(1)}} \cdot \dfrac{\partial y^{(1)}}{\partial w_{i,j}^{(1)}} \\ \dfrac{\partial \mathcal{L}}{\partial b_i^{(1)}} = \dfrac{\partial \mathcal{L}}{\partial z} \cdot \dfrac{\partial z}{\partial x_*^{(3)}} \cdot \dfrac{\partial x_*^{(3)}}{\partial y^{(2)}} \cdot \dfrac{\partial y^{(2)}}{\partial x^{(2)}} \cdot \dfrac{\partial x^{(2)}}{\partial y^{(1)}} \cdot \dfrac{\partial y^{(1)}}{\partial b_i^{(1)}} \end{cases}$$

Backward propagation

From the results in previous pages, we can have

$$\frac{\partial \mathcal{L}}{\partial w_{i,j}^{(1)}} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial x_*^{(3)}} \cdot \frac{\partial x_*^{(3)}}{\partial y^{(2)}} \cdot \frac{\partial y^{(2)}}{\partial x^{(2)}} \cdot \frac{\partial x^{(2)}}{\partial y^{(1)}} \cdot \frac{\partial y^{(1)}}{\partial w_{i,j}^{(1)}}$$

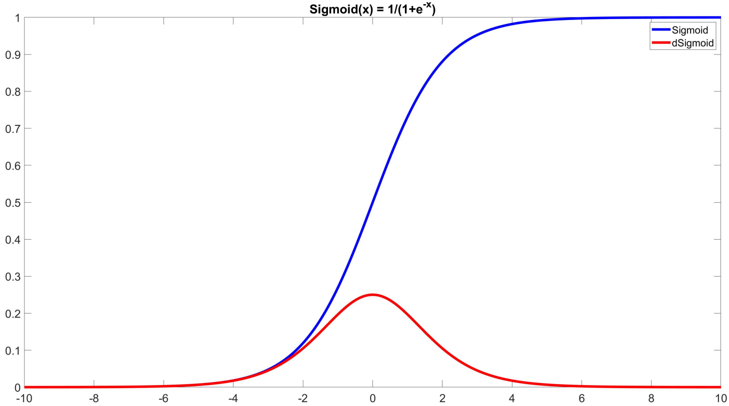
$$= \frac{\partial \mathcal{L}}{\partial z} \cdot \left[\left(w_1^{(3)} \cdot \sigma_2'(y_1^{(2)}) \cdot w_{1,1}^{(2)} + w_2^{(3)} \cdot \sigma_2'(y_2^{(2)}) \cdot w_{1,2}^{(2)} \right) \cdot \sigma_1'(y_1^{(1)}) \cdot x_1 \right]$$

$$\frac{\partial \mathcal{L}}{\partial b_i^{(1)}} = \frac{\partial \mathcal{L}}{\partial z} \cdot \frac{\partial z}{\partial x_*^{(3)}} \cdot \frac{\partial x_*^{(3)}}{\partial y^{(2)}} \cdot \frac{\partial y^{(2)}}{\partial x^{(2)}} \cdot \frac{\partial x^{(2)}}{\partial y^{(1)}} \cdot \frac{\partial y^{(1)}}{\partial b_i^{(1)}}$$

$$= \frac{\partial \mathcal{L}}{\partial z} \cdot \left[\left(w_1^{(3)} \cdot \sigma_2'(y_1^{(2)}) \cdot w_{1,1}^{(2)} + w_2^{(3)} \cdot \sigma_2'(y_2^{(2)}) \cdot w_{1,2}^{(2)} \right) \cdot \sigma_1'(y_1^{(1)}) \cdot 1 \right]$$

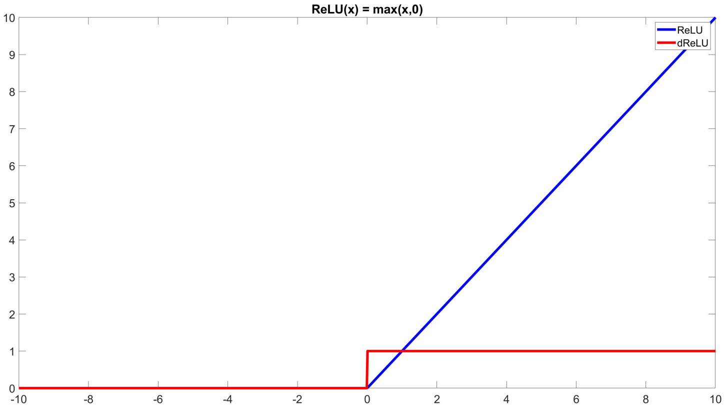
Finally, we can update the weights and biases by previous optimization method.

Revisit activation functions



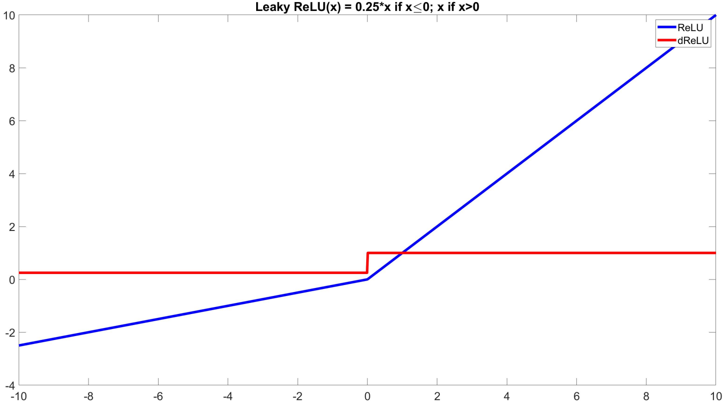
Sigmoid

Revisit activation functions



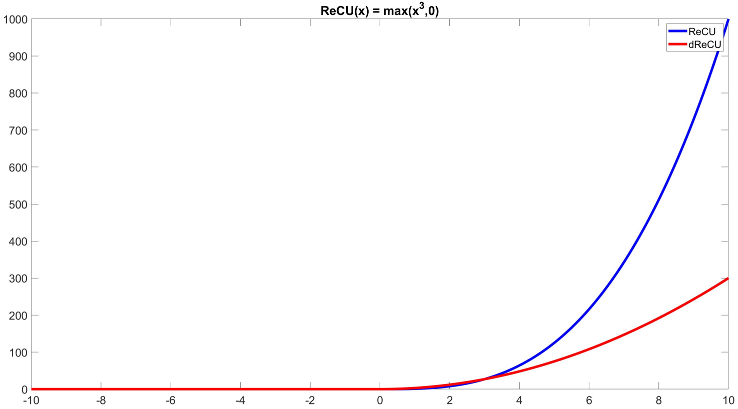
ReLU

Revisit activation functions



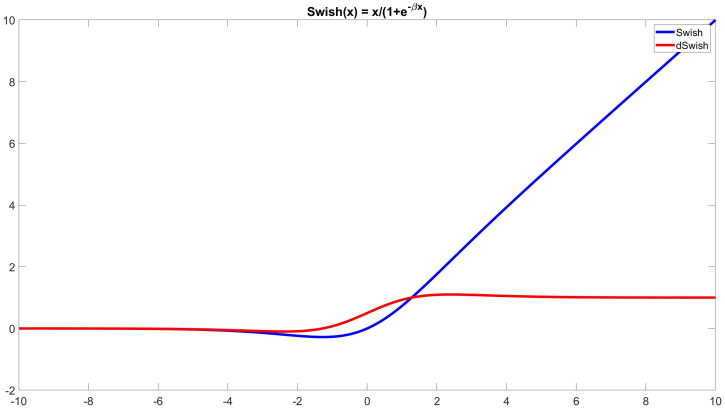
Leaky ReLU

Revisit activation functions



ReCU

Revisit activation functions



Swish

Poisson equation

Consider the Poisson equation with Dirichlet boundary conditions

$$\begin{cases} \Delta u = f, & \text{in } \Omega, \\ u = g, & \text{on } \partial\Omega \end{cases}$$

We implement following methods to solve the Poisson equation

- Deep Galerkin Method (DGM)
- Deep Ritz Method (DRM)

Deep Galerkin Method (DGM)

Loss function:

$$\begin{aligned}\mathcal{L}[u] &= \|\Delta u - f\|_{2,\Omega}^2 + \lambda \|u - g\|_{2,\partial\Omega}^2 \\ &= \int_{\Omega} (\Delta u - f)^2 dx + \lambda \int_{\partial\Omega} (u - g)^2 dx\end{aligned}$$

Goal:

$$\min_{u \in \mathcal{F}} \mathcal{L}[u]$$

where $\mathcal{F}$ is the class of neural networks.

Monte Carlo approach

Monte Carlo approach

$$I := \int_a^b f(x)dx = (b-a)\int_a^b f(x)\cdot\frac{1}{b-a}dx = (b-a)\mathbb{E}[f(X)]$$

where $X \sim U(a,b)$.

1. Generate $X_1, ..., X_N \overset{iid}{\sim} U(a,b)$
2. Compute $\hat{I}_N = \frac{b-a}{N}\sum_{i=1}^{N} f(X_i)$

Monte Carlo approach

- **Unbiased estimation:**

$$\mathbb{E}[\hat{I}_N] = \mathbb{E}\left[\frac{b-a}{N}\sum_{i=1}^{N} f(X_i)\right] = \frac{1}{N}\sum_{i=1}^{N}(b-a)\mathbb{E}\left[f(X_i)\right] = I$$

- **Probability convergence:**
  By Law of Large Number, for any $\varepsilon > 0$, there exists $N \in \mathbb{N}$ such that

$$\mathbb{P}(|\hat{I}_N - I| > \varepsilon) = 0$$

- **Convergent rate:** By Center Limit Theorem,

$$\frac{\hat{I}_N - I}{\frac{\sigma}{\sqrt{N}}} \xrightarrow{\mathcal{D}} \mathcal{N}(0,1)$$

  where σ is population standard deviation. The error convergence rate is $\mathcal{O}(\frac{1}{\sqrt{N}})$.

Deep Galerkin Method (DGM)

$$\mathcal{L}[u] = |\Omega|\mathbb{E}_{x\sim p(x)}[(\Delta u(x) - f(x))^2] + \lambda|\partial\Omega|\mathbb{E}_{x\sim q(x)}[(u(x) - g(x))^2]$$

where $p(x)$ is a uniform distribution on Ω and $q(x)$ is a uniform distribution on $\partial\Omega$.

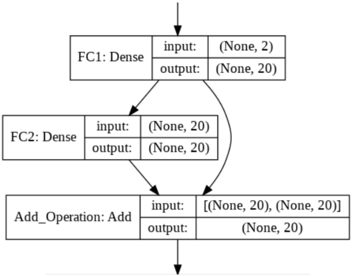
$$\mathcal{L}[u] = \frac{|\Omega|}{N}\sum_{i=1}^{N}[\Delta u(x_i) - f(x_i)]^2 + \lambda\frac{|\partial\Omega|}{M}\sum_{j=1}^{M}[u(t_j) - g(t_j)]^2$$

where $x_i \in \Omega$ and $t_j \in \partial\Omega$, for all $i = 1, 2, ...N$, $j = 1, 2, ..., M$.

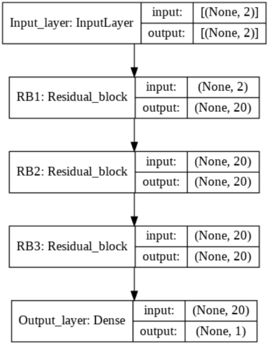
Numerical result of DGM

Information:

- Network: ResNet
- Activation function: Swish



Residual block structure



Model structure

Numerical result of DGM

Information (continue):

Residual block (RB1)

Layer	Input shape	Output shape	parameters
FC1	(batch size, 2)	(batch size, 20)	60
FC2	(batch size, 20)	(batch size, 20)	420

ResNet model

Layer	Input shape	Output shape	parameters
RB1	(batch size, 2)	(batch size, 20)	480
RB2	(batch size, 20)	(batch size, 20)	840
RB3	(batch size, 20)	(batch size, 20)	840
Output layer	(batch size, 20)	(batch size, 1)	21

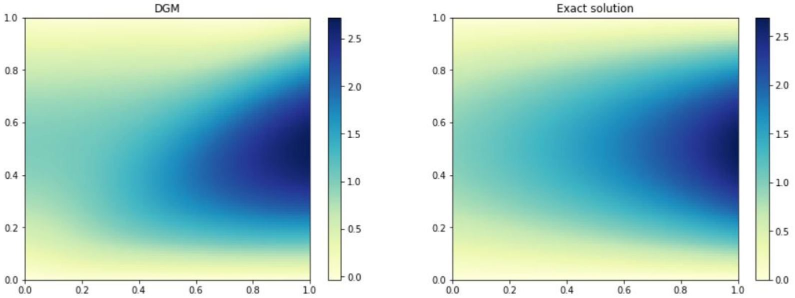
- Total parameters : 2181

Numerical result of DGM

Information (continue):

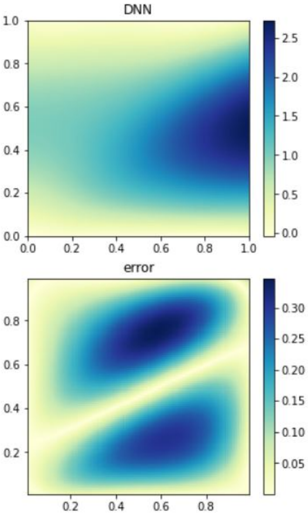
- Exact solution: $u = e^x \sin(\pi y)$
- Epochs: 20000
- Learning rate: $5e - 4$
- Penalty term: $\lambda = 1$
- Number of training points: 110 (interior: 100 / boundary: 10)
- Number of testing points: 10000 (uniform mesh by 100×100)
- Device: Google Colab (GPU accelerated)
- Total time: 1200s (0.06 s/ep)

Numerical result of DGM

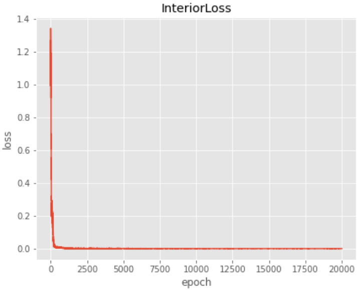


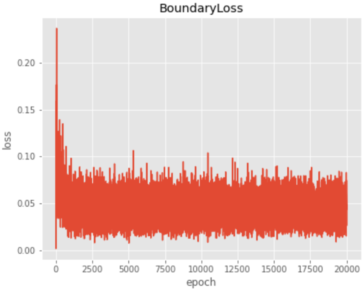
uniform mesh by 100×100

Numerical result of DGM

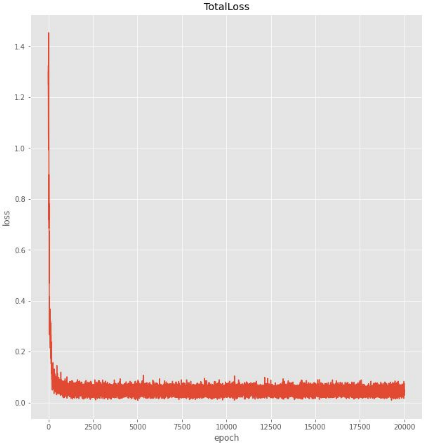


Numerical result of DGM





Numerical result of DGM



Numerical result of DGM

- Number of testing points: 100×100

error ╲ epoch	5000	10000	20000
$\|U-u\|_\infty$	0.3253	0.3461	0.3668
$\|U-u\|_2$	0.1633	0.1657	0.1725
$\frac{\|U-u\|_2}{\|u\|_2}$	0.1296	0.1315	0.1369

Deep Ritz Method (DRM)

Loss function:

$$\mathcal{L}[u] = \int_{\Omega} \left(\frac{1}{2} |\nabla u|^2 + f u \right) dx + \lambda \int_{\partial\Omega} (u - g)^2 dx$$

Goal:

$$\min_{u \in \mathcal{F}} \mathcal{L}[u]$$

where $\mathcal{F}$ is the class of neural networks.

Energy functional

Consider the functional

$$\mathcal{J}[v] = \int_\Omega \left(\frac{1}{2}|\nabla v|^2 + fv\right) dx =: \int_\Omega F[v]dx.$$

Suppose $\mathcal{J}[v]$ has local minimum at u. Then for any $w \in \mathcal{C}_0^\infty(\Omega)$, we have

$$\mathcal{J}[u] \leq \mathcal{J}[u + \varepsilon w]$$

as ε closed to 0. Define $\Phi(\varepsilon) = \mathcal{J}[u + \varepsilon w]$. Then

$$\Phi'(0) = \frac{d\Phi(\varepsilon)}{d\varepsilon}\bigg|_{\varepsilon=0} = \int_\Omega \frac{dF[u + \varepsilon w]}{d\varepsilon}\bigg|_{\varepsilon=0} dx = 0$$

Energy functional

Note that

$$F[u+\varepsilon w] = F[u] + \frac{1}{2}\varepsilon^2|\nabla w|^2 + \varepsilon \nabla u \cdot \nabla w + \varepsilon f w$$

Then

$$\Phi'(0) = \int_\Omega \left(\varepsilon|\nabla w|^2 + \nabla u \cdot \nabla w + f w\right)\Big|_{\varepsilon=0} dx = 0$$

that is,

$$\int_\Omega (\nabla u \cdot \nabla w + f w)\, dx = 0$$

Energy functional

Green's first identity

$$\int_{\Omega} \Delta u w dx = \int_{\partial\Omega} \frac{\partial u}{\partial n} \cdot w ds - \int_{\Omega} \nabla u \cdot \nabla w dx$$

Since $w \in \mathcal{C}_0^\infty(\Omega)$,

$$\int_{\Omega} (-\Delta u + f)\, w dx = 0$$

Hence we can get

$$\Delta u = f.$$

Deep Ritz Method (DRM)

$$\mathcal{L}[u] = |\Omega|\mathbb{E}_{x\sim p}\left[\frac{1}{2}|\nabla u(x)|^2 + f(x)u(x)\right] + \lambda|\partial\Omega|\mathbb{E}_{x\sim q}[(u(x) - g(x))^2]$$

where $p(x)$ is a uniform distribution on Ω and $q(x)$ is a uniform distribution on $\partial\Omega$.

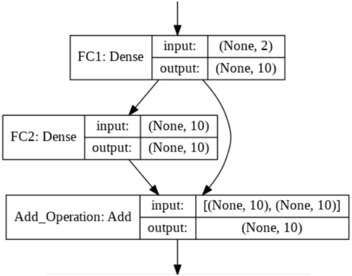
$$\mathcal{L}[u] = \frac{|\Omega|}{N}\sum_{i=1}^{N}\left[\frac{1}{2}|\nabla u(x_i)|^2 + f(x_i)u(x_i)\right] + \lambda\frac{|\partial\Omega|}{M}\sum_{j=1}^{M}[u(t_j) - g(t_j)]^2$$

where $x_i \in \Omega$ and $t_j \in \partial\Omega$, for all $i = 1, 2, ...N$, $j = 1, 2, ..., M$.

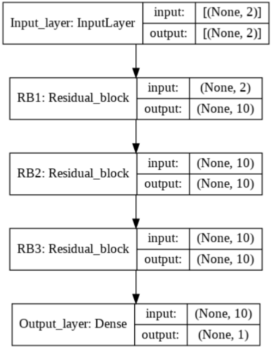
Numerical result of DRM

Information:

- Network: ResNet
- Activation function: ReCU



Residual block structure



Model structure

Numerical result of DRM

Information (continue):

Residual block (RB1)

Layer	Input shape	Output shape	parameters
FC1	(batch size, 2)	(batch size, 10)	30
FC2	(batch size, 10)	(batch size, 10)	110

ResNet model

Layer	Input shape	Output shape	parameters
RB1	(batch size, 2)	(batch size, 10)	140
RB2	(batch size, 10)	(batch size, 10)	220
RB3	(batch size, 10)	(batch size, 10)	220
Output layer	(batch size, 10)	(batch size, 1)	11

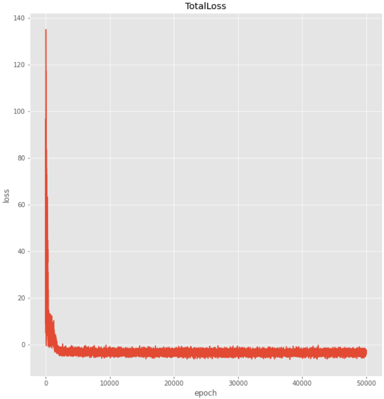
- Total parameters : 591

Numerical result of DRM

Information (continue):

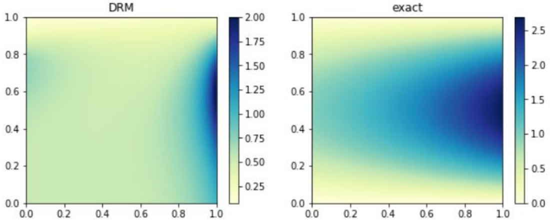
- Exact solution: $u = e^x \sin(\pi y)$
- Epochs: 20000
- Learning rate: $5e - 4$
- Penalty term: $\lambda = 5000$
- Number of training points: 600 (interior: 500 / boundary: 100)
- Number of testing points: 10000 (uniform mesh by 100×100)
- Device: Google Colab (GPU accelerated)
- Total time: 400s (0.02 s/ep)

Numerical result for DRM

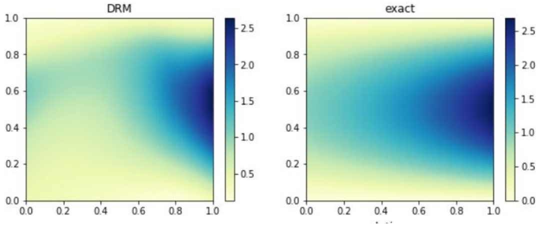


Numerical result of DRM

epoch 1000

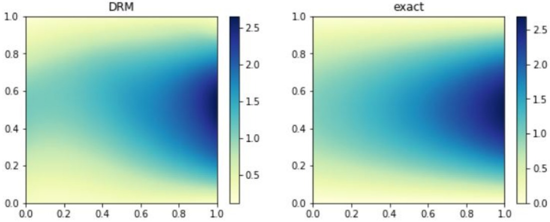


epoch 2500

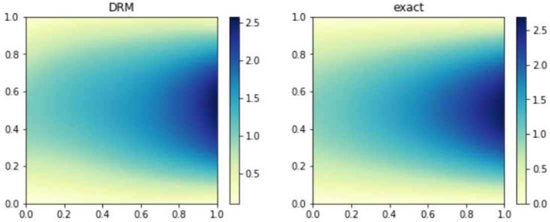


Numerical result of DRM

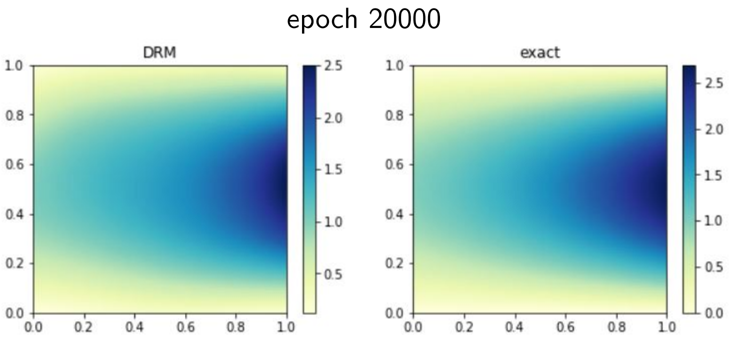
epoch 5000



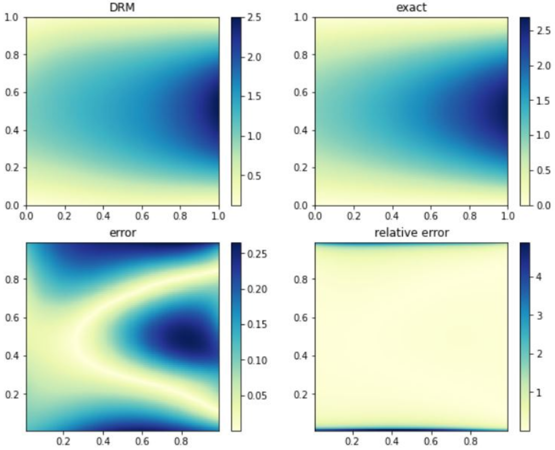
epoch 10000



Numerical result of DRM



Numerical result



Numerical result of DRM

- Number of testing points: 100×100

error ╲ epoch	5000	10000	20000
$\|U-u\|_\infty$	0.3329	0.2908	0.2911
$\|U-u\|_2$	0.1128	0.1112	0.1233
$\frac{\|U-u\|_2}{\|u\|_2}$	0.0896	0.0883	0.0979

Biharmonic equation

Consider the Biharmonic equation with boundary conditions

$$\begin{cases} \Delta^2 u = f, & \text{in } \Omega, \\ u = g_0, & \text{on } \partial\Omega, \\ \dfrac{\partial u}{\partial n} = g_1, & \text{on } \partial\Omega \end{cases}$$

To make the calculation easier, we rewrite the equation as following,

$$\begin{cases} \Delta u = p, & \text{in } \Omega, \\ \Delta p = f, & \text{in } \Omega, \\ u = g_0, & \text{on } \partial\Omega, \\ \dfrac{\partial u}{\partial n} = g_1, & \text{on } \partial\Omega \end{cases}$$

DGM for biharmonic equation

Loss function:

$$\begin{aligned}\mathcal{L}[u] &= \|\Delta u - p\|^2_{2,\Omega} + \|\Delta p - f\|^2_{2,\Omega} \\ &\quad + \alpha\|u - g_0\|^2_{2,\partial\Omega} + \beta\|(\nabla u \cdot n) - g_1\|^2_{2,\partial\Omega}\end{aligned}$$

By Monte Carlo approach,

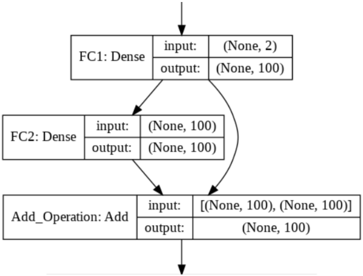
$$\begin{aligned}\mathcal{L}[u] &= \frac{|\Omega|}{N}\sum_{i=1}^{N}[\Delta u(x_i) - p(x_i)]^2 + \frac{|\Omega|}{N}\sum_{i=1}^{N}[\Delta p(x_i) - f(x_i)]^2 \\ &\quad + \alpha\frac{|\partial\Omega|}{M}\sum_{j=1}^{M}[u(t_j) - g_0(t_j)]^2 + \beta\frac{|\partial\Omega|}{M}\sum_{j=1}^{N}[\nabla u(t_j)\cdot n(t_j) - g_1(t_j)]^2\end{aligned}$$

where $x_i \in \Omega$ and $t_j \in \partial\Omega$, for all $i = 1, 2, ...N$, $j = 1, 2, ..., M$.

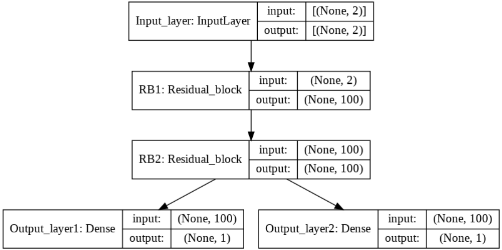
Numerical result of DGM

Information:

- Network: ResNet
- Activation function: Swish



Residual block structure



Model structure

Numerical result of DGM

Information (continue):

Residual block (RB1)

Layer	Input shape	Output shape	parameters
FC1	(batch size, 2)	(batch size, 100)	300
FC2	(batch size, 100)	(batch size, 100)	10100

ResNet model

Layer	Input shape	Output shape	parameters
RB1	(batch size, 2)	(batch size, 100)	10400
RB2	(batch size, 100)	(batch size, 100)	20200
Output layer1	(batch size, 100)	(batch size, 1)	101
Output layer2	(batch size, 100)	(batch size, 1)	101

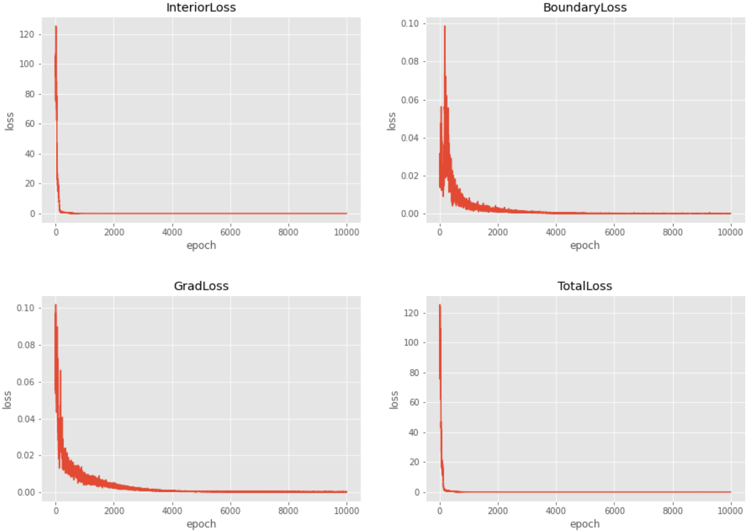
- Total parameters : 30802

Numerical result of DGM

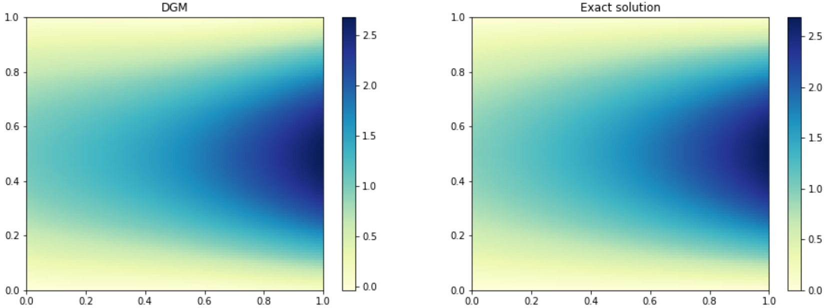
Information (continue):

- Exact solution: $u = e^x \sin(\pi y)$
- Epochs: 10000
- Learning rate: $5e - 4$
- Penalty term: $\lambda = 1$
- Number of training points: 130 (interior: 100 / boundary: 30)
- Number of testing points: 10000 (uniform mesh by 100×100)
- Device: Google Colab (GPU accelerated)
- Total time: 1040s (0.1 s/ep)

Numerical result of DGM

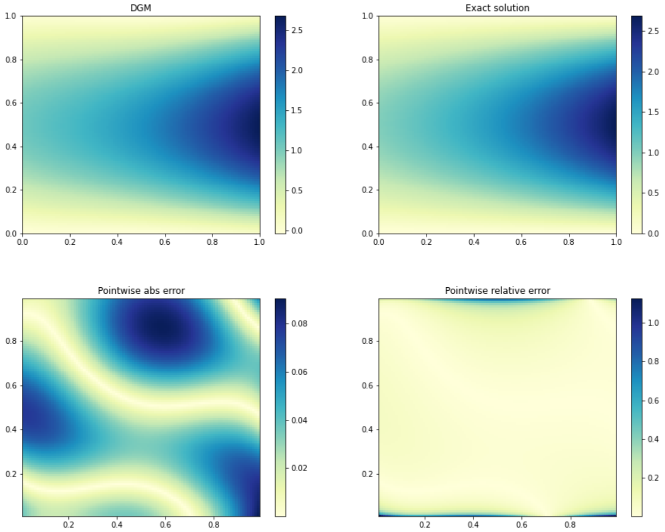


Numerical result of DGM



uniform mesh by 100×100

Numerical result of DGM



Numerical result of DGM

- Number of training points: $100 + 30$ / ep
- Number of testing points: 100×100

error ╲ epoch	2000	4000	6000	8000	10000
$\|U - u\|_\infty$	0.4190	0.1517	0.1456	0.1085	0.0993
$\|U - u\|_2$	0.1419	0.0701	0.0542	0.0519	0.0452
$\frac{\|U - u\|_2}{\|u\|_2}$	0.1126	0.0556	0.0430	0.0412	0.0359

Numerical result of DGM

- Number of testing points: 100×100
- Error: relative error of two norm

Tp \ epoch	2000	4000	6000	8000	10000
100/10	0.0619	0.0426	0.0408	0.0421	0.0391
400/20	0.0417	0.0316	0.0357	0.0331	0.0270
900/30	0.0363	0.0257	0.0347	0.0278	0.0264

Numerical result of DGM

- Epochs: 10000
- Number of training points: $100 + 30$ / ep
- Number of testing points: 100×100

error	Swish	Sigmoid	ReLU
$\|U-u\|_\infty$	0.0993	0.2292	0.0723
$\|U-u\|_2$	0.0452	0.0897	0.0220
$\frac{\|U-u\|_2}{\|u\|_2}$	0.0359	0.0712	0.0175

Code on Github

- Poisson DGM:
  https://github.com/Jia-wei-liao/NPDE_final_project/blob/main/DGM_Poisson2D.ipynb
- Possion DRM:
  https://github.com/Jia-wei-liao/NPDE_final_project/blob/main/DRM_Poisson2D.ipynb
- Biharmonic DGM:
  https://github.com/Jia-wei-liao/NPDE_final_project/blob/main/DGM_Biharmonic2D.ipynb

References

- Jingrun Chen, Rui Du and Keke Wu, A Comparison Study of Deep Galerkin Method and Deep Ritz Method for Elliptic Problems with Different Boundary Conditions (2020).
- Liyao Lyu, Zhen Zhangc, Minxin Chen, Jingrun Chen, MIM: A deep mixed residual method for solving high-order partial differential equations (2020).
- Weinan E and Bing Yu, The Deep Ritz method: A deep learning-based numerical algorithm for solving variational problems (2017).

Thanks for listening!