Solving PDEs with Neural Network

Zunding Huang

UC San Diego

04/17/2023

Outline

1. Deep Ritz method, Physics-informed neural networks (PINN)
2. Implementation and results

Solving PDEs with Neural Network

Deep Ritz method

- Ritz Galerkin method: A direct method to find an approximate solution for boundary value problems, where the differential equation for a physical system can be formulated via minimization of a quadratic function representing the system energy and the approximate solution is a linear combination of the given set of the basis functions.
- Deep Ritz method: Deep NN + Ritz method, for solving variational problem.

If we want to solve the following Possion's equation:

$$\begin{cases} -\Delta \mathrm{u} = \mathrm{f} \text{ in } \Omega, \\ \mathrm{u} = \mathrm{g} \text{ in } \partial\Omega. \end{cases}$$

Solving PDEs with Neural Network

Deep Ritz method

It is equivalent to

$$\min_{u \in H} \ I(u)$$

where

$$I(u) = \int_{\Omega} \left(\frac{1}{2} |\nabla u(x)|^2 - f(x)u(x) \right) dx$$

and

$$H = \{u \in H^1(\Omega) : u = g \text{ on } \partial\Omega\}$$

Deep Ritz method is based on:

- 1. Deep neural network based approximation of the trial function
- 2. A numerical quadrature rule for the functional
- 3. An algorithm for solving the final optimization problem

Solving PDEs with Neural Network

Building trial function

The basic component of DR method is a nonlinear transformation

$$\mathrm{x} \in \mathbb{R}^{\mathrm{n}} \ \rightarrow \mathrm{z}_{\theta}(\mathrm{x}) \in \mathbb{R}^{\mathrm{m}}$$

defined by a deep neural network. Here θ represents parameters, typically weights and biases in the neural network.
In the paper, the author uses an architecture that each layer of the network is constructed by stacking several blocks. The i-th block can be expressed by

$$\mathrm{t} = \mathrm{f_i}(\mathrm{s}) = \phi\left(\mathrm{W_{i,2}} \cdot \phi\left(\mathrm{W_{i,1}s} + \mathrm{b_{i,1}}\right) + \mathrm{b_{i,2}}\right) + \mathrm{s}$$

where $\mathrm{W_{i,1}}, \mathrm{W_{i,2}} \in \mathbb{R}^{\mathrm{m}\times\mathrm{m}}, \mathrm{b_{i,1}}, \mathrm{b_{i,2}} \in \mathbb{R}^{\mathrm{m}}$ and ϕ is the activiation function.

Solving PDEs with Neural Network

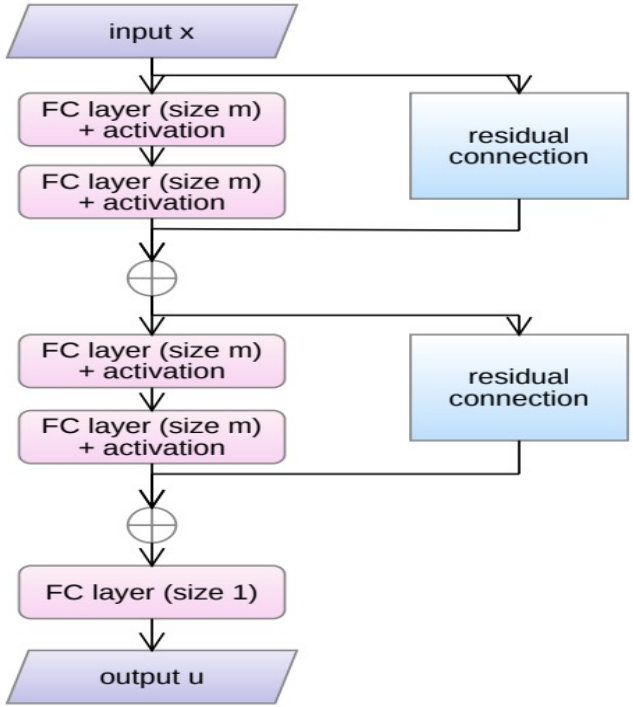

Figure 1: A network with two blocks and an output linear layer. Each block consists of two fully-connected layers and a skip connection.

Solving PDEs with Neural Network

Building trial function

The full n-block network can be expressed as

$$\mathrm{z}_\theta(\mathrm{x}) = \mathrm{f_n} \circ \mathrm{f_{n-1}} ... \circ \mathrm{f_1}(\mathrm{x})$$

where θ represents all the parameters in the neural network.
We should be careful here since the input x is in $\mathbb{R}^d$ instead of $\mathbb{R}^m$ so we should apply a linear transform on x.
Once having $\mathrm{z}_\theta(\mathrm{x})$, we obtain u by

$$\mathrm{u}(\mathrm{x};\theta) = \mathrm{a} \cdot \mathrm{z}_\theta(\mathrm{x}) + \mathrm{b}$$

Solving PDEs with Neural Network

Building trial function

Denote

$$\mathrm{h}(\mathrm{x};\theta) = \frac{1}{2}|\nabla_{\mathrm{x}}\mathrm{u}(\mathrm{x};\theta)|^2 - \mathrm{f}(\mathrm{x})\mathrm{u}(\mathrm{x};\theta)$$

Then original problem

$$\begin{cases} \min_{\mathrm{u}\in\mathrm{H}} \ \mathrm{I}(\mathrm{u}), \\ \mathrm{I}(\mathrm{u}) = \int_\Omega \left(\frac{1}{2}|\nabla \mathrm{u}(\mathrm{x})|^2 - \mathrm{f}(\mathrm{x})\mathrm{u}(\mathrm{x})\right) \mathrm{dx} \end{cases}$$

will be converted to a numerical optimization problem:

$$\begin{cases} \min_{\theta} \ \mathrm{L}(\theta), \\ \mathrm{L}(\theta) = \int_\Omega \mathrm{h}(\mathrm{x};\theta)\mathrm{dx} \end{cases}$$

Solving PDEs with Neural Network

Building trial function

Since u belongs to admissble set H, where

$$\mathrm{H} = \{\mathrm{u} \in \mathrm{H}^1(\Omega) : \mathrm{u} = \mathrm{g} \text{ on } \partial\Omega\}$$

In real, we will use a penalty method and the numerical optimization problem should be:

$$\begin{aligned}\min_{\theta} \ \mathrm{L}(\theta), \mathrm{L}(\theta) &= \int_{\Omega} \mathrm{h}(\mathrm{x};\theta)\mathrm{dx} + \beta \int_{\partial\Omega} (\mathrm{u}-\mathrm{g})^2 \mathrm{dx} \\ &\approx \int_{\Omega} \left(\frac{1}{2} |\nabla_{\mathrm{x}} \mathrm{u}(\mathrm{x};\theta)|^2 - \mathrm{f}(\mathrm{x})\mathrm{u}(\mathrm{x};\theta) \right) \mathrm{dx} + \\ &\quad \beta \int_{\partial\Omega} (\mathrm{u}(\mathrm{x};\theta) - \mathrm{g})^2 \mathrm{dx}\end{aligned}$$

where β is the penalty coefficient.

Solving PDEs with Neural Network

Stochastic gradient descent and numerical quadrature rule

In deep learning, the optimization problem often takes the form of:

$$\min_{\theta} \ \mathrm{L}(\theta), \mathrm{L}(\theta) = \frac{1}{\mathrm{N}} \sum_{\mathrm{i}=1}^{\mathrm{N}} \mathrm{L_i}(\theta)$$

where each $\mathrm{L_i}(\theta)$ corresponds to a data point and N is typically very large. Some optimization methods used in deep learning:

- 1. Gradient Descent
- 2. Stochastic Gradient Descent
- 3. Adam, an updated version of SGD

Solving PDEs with Neural Network

Stochastic gradient descent and numerical quadrature rule

- At each step of the SGD iteration, one chooses a mini-batch of points to discretize the integral. These points are chosen randomly and the same quadrature weight is used at every point.
- Notice: If we use standard quadrature rules to discretize the integral, then we are bound to choose a fixed set of nodes. In this case, we run into the risk where the integrand is minimized on these fixed nodes but the functional itself is far from being minimized.

Solving PDEs with Neural Network

Stochastic gradient descent and numerical quadrature rule

SGD in this context is given by

$$\theta^{k+1} = \theta^{k} - \eta \nabla_{\theta} \frac{1}{N} \sum_{j=1}^{N} h(x_{j,k}; \theta^{k})$$

where $\{x_{j,k}\}$ is a set of points in Ω that are randomly sampled with uniform distribution.

Question: How to calculate the gradient with respect to x?

Answer: Just use finite difference scheme to approximate.

Solving PDEs with Neural Network

Numerical results

The first problem is Possion's equation in two dimension:
$\Omega = [-1, 1] \times [-1, 1]$

$$\begin{cases} -\Delta \mathrm{u}(\mathrm{x}, \mathrm{y}) = 1 \text{ in } \Omega, \\ \mathrm{u}(\mathrm{x}, \mathrm{y}) = -(\mathrm{x}^2 + \mathrm{y}^2)/4 \text{ on } \partial\Omega. \end{cases}$$

The true solution to this problem is

$$\mathrm{u}(\mathrm{x}, \mathrm{y}) = -(\mathrm{x}^2 + \mathrm{y}^2)/4.$$

in whole region Ω.
We choose penalty coefficient $\beta = 500$, the number of grid points in Ω is $N_1 = 600$ and the number of grid points on $\partial\Omega$ is $N_2 = 60$.

Solving PDEs with Neural Network

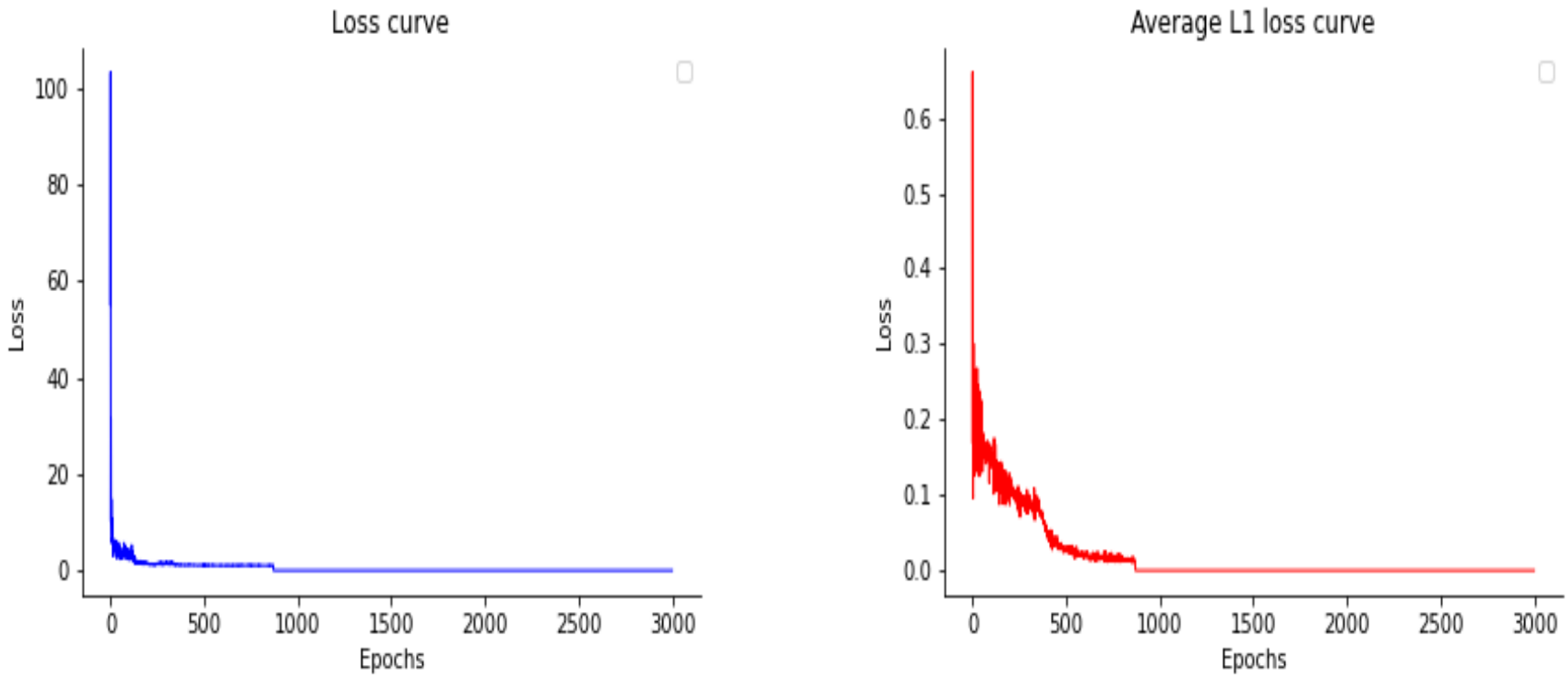

Figure 2: (Left) The value of loss vs. the number of iterations.
(Right) The relative L^1-error between numerical solution and true solution vs. the number of iterations.

Solving PDEs with Neural Network

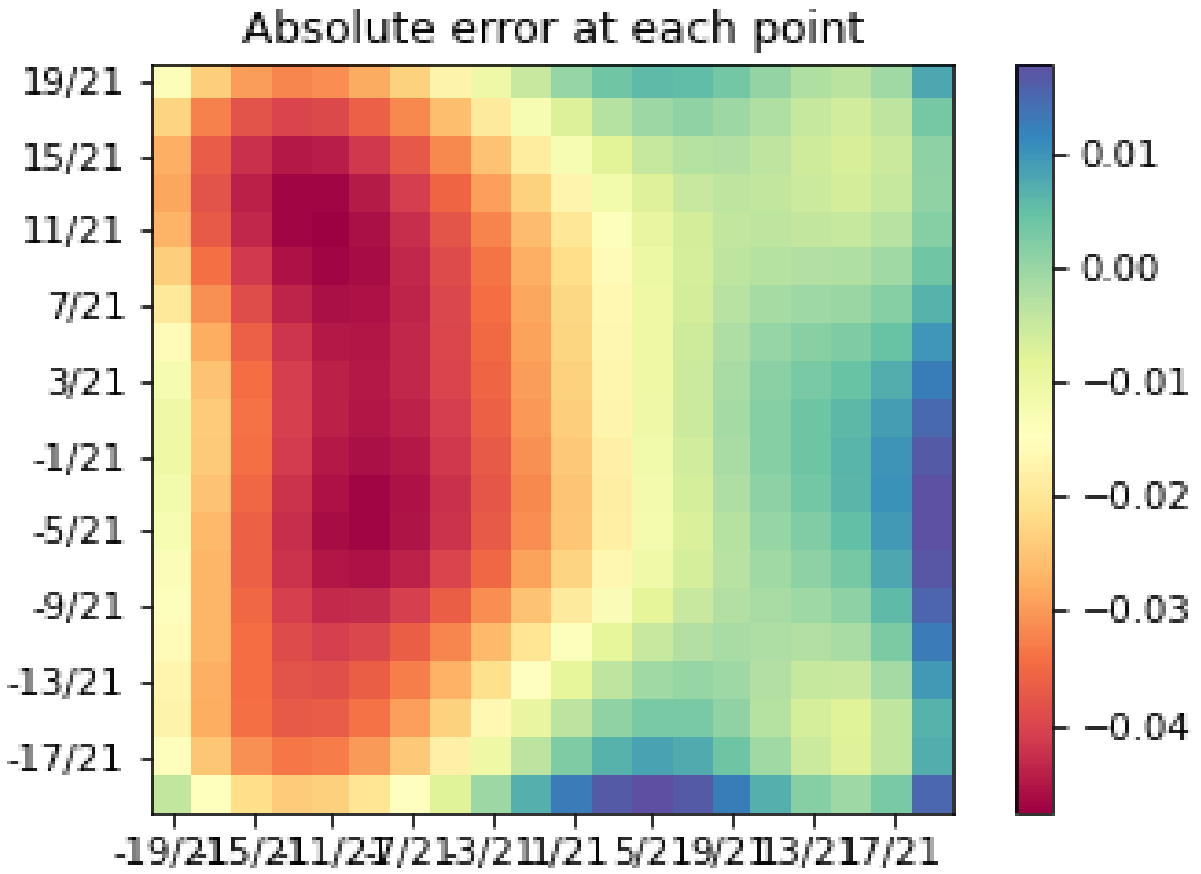

Figure 3: The absolute error between numerical solution and true solution in the whole computational region Ω.

Solving PDEs with Neural Network

Numerical results

The second problem is Possion's equation in $\mathbb{R}^{10}$:

$$\begin{cases} \Delta u(x) = 0 \text{ in } \Omega, \\ u(x) = x_1x_2 + x_3x_4 + x_5x_6 + x_7x_8 + x_9x_{10} \text{ on } \partial\Omega. \end{cases}$$

where

$$\Omega = \{x \in \mathbb{R}^{10} : \|x\| < 1\}.$$

The true solution to this problem is

$$u(x) = x_1x_2 + x_3x_4 + x_5x_6 + x_7x_8 + x_9x_{10}.$$

in whole region Ω.
We choose penalty coefficient $\beta = 500$, the number of grid points in Ω is $N_1 = 1000$ and the number of grid points on $\partial\Omega$ is $N_2 = 2000$.

Solving PDEs with Neural Network

This is the experiment for $\mathrm{N} = 5000$.

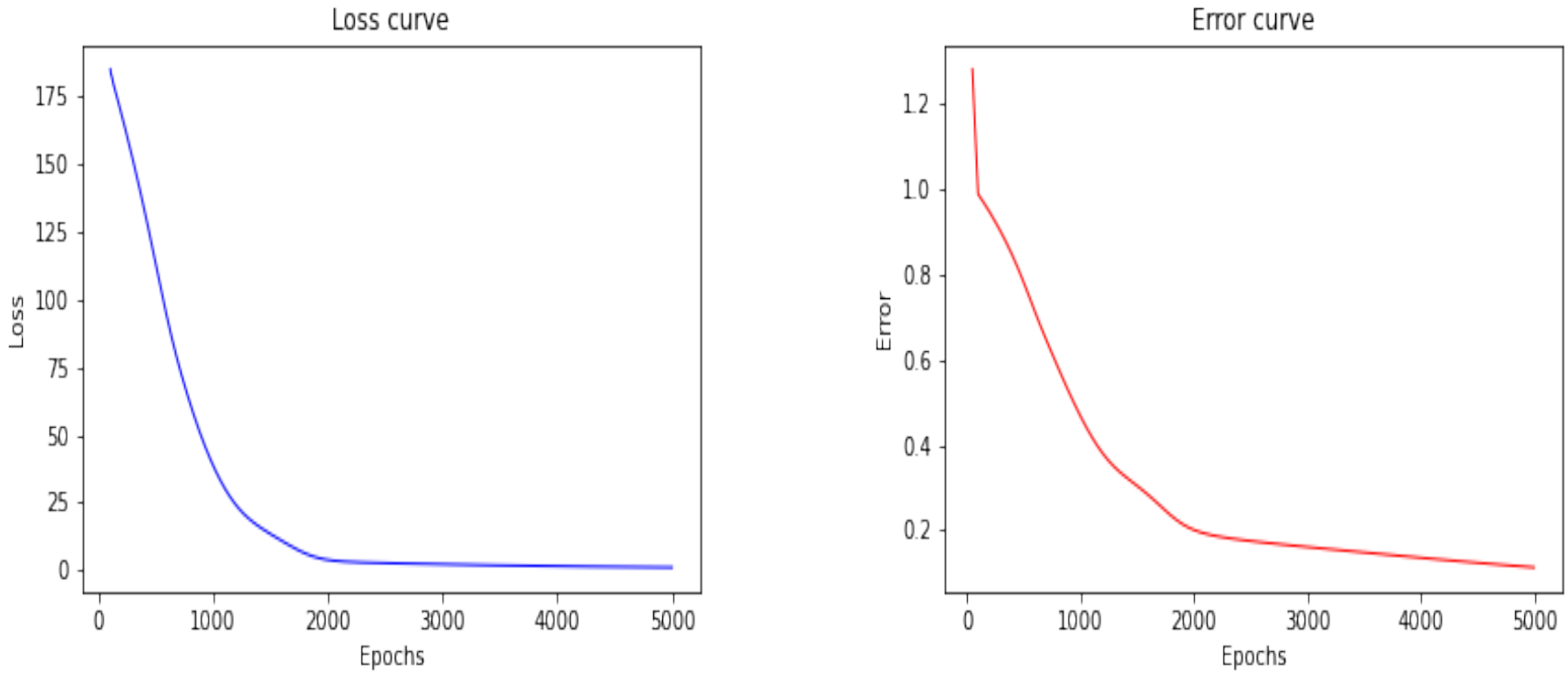

Figure 4: (Left) The value of loss vs. the number of iterations.
(Right) The relative L^2-error between numerical solution and true solution vs. the number of iterations.

Solving PDEs with Neural Network

This is the experiment for $\mathrm{N} = 100,000$.

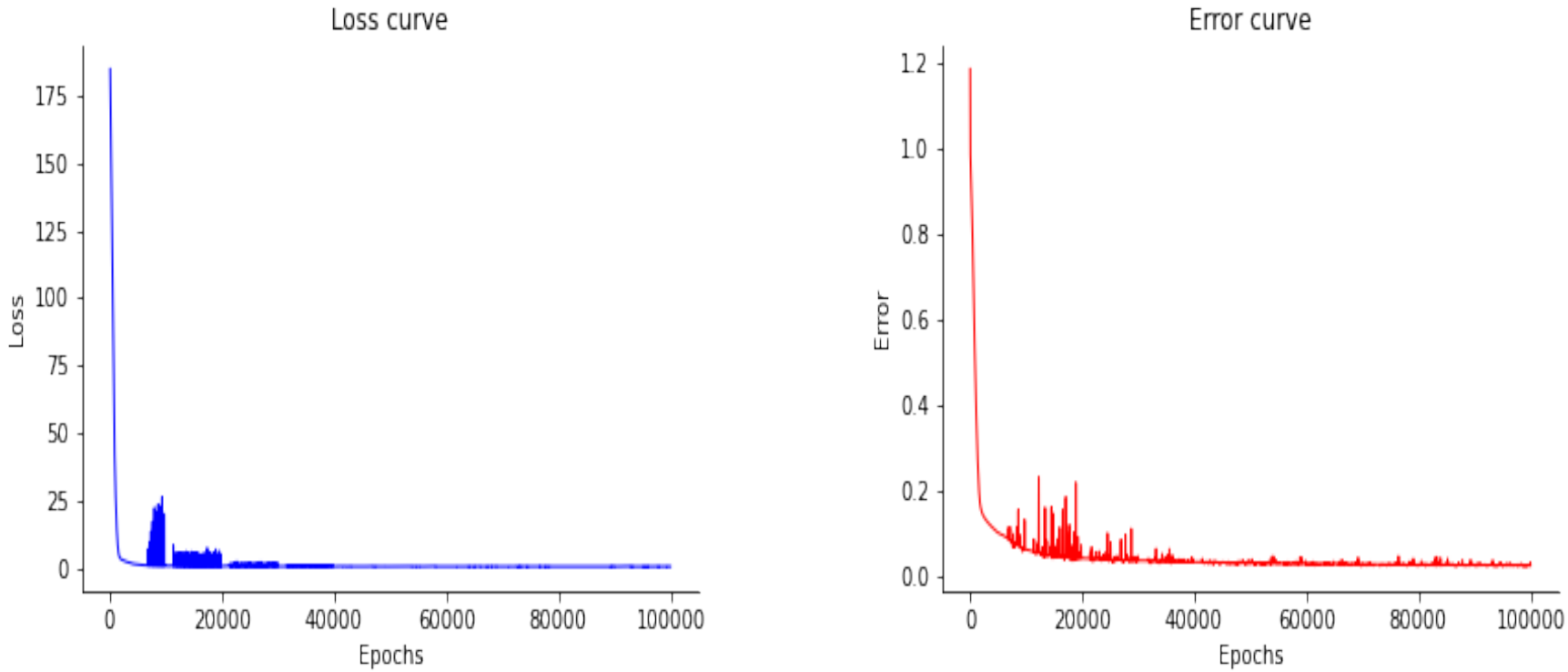

Figure 5: (Left) The value of loss vs. the number of iterations.
(Right) The relative L^2-error between numerical solution and true solution vs. the number of iterations.

Solving PDEs with Neural Network

Numerical results

For the experiment with $\mathrm{N} = 100,000$, we have

- The L2 relative error on the test set is 0.0254205,
- Training costs 3002 seconds,
- Whole model has 561 parameters.

Solving PDEs with Neural Network

Conclusion

Advantages:

- It is less sensitive to the dimensionality of the problem and has the potential to work in rather high dimensions.
- The method is reasonably simple and fits well with the stochastic gradient descent framework commonly used in deep learning.

Future Work:

- The variational problem that we obtain at the end is not convex. The issue of local minima and saddle points is non-trivial.
- There is no consistent conclusion about the convergence rate.
- The treatment of the essential boundary condition is not as simple as for the traditional methods.

Solving PDEs with Neural Network

Physics Informed Neural Network

- PINNs - Neural networks that are trained to solve supervised learning tasks while respecting physical laws (PDEs)
- Identify a nonlinear map from a few, potentially very high dimensional input and output pairs of data
- However, many physical and biological systems consist of prior knowledge encoded in physical laws
- This prior information can act as a constraint that reduces the space of admissible solutions, remove unrealistic solutions that violate fundamental conservation laws

Solving PDEs with Neural Network

Physics Informed Neural Network

Consider following PDEs:

$$\mathrm{u_t} + \mathscr{N}(\mathrm{u};\ \lambda) = 0$$

- Data-driven solution — When λ is a known, how to find the unknown solution $\mathrm{u}(\mathrm{t}, \mathrm{x})$?
- Data-driven discovery of PDEs — When λ is an unknown, how to find the solution $\mathrm{u}(\mathrm{t}, \mathrm{x})$ and fix λ simultaneously?

PDEs:

- Need both initial conditions and boundary conditions
- Point Collocation methods: Function Approximation + point evaluation

Solving PDEs with Neural Network

Physics Informed Neural Network

- Parameterized, nonlinear PDE(s)

$$u_t + \mathcal{N}(u;\ \lambda) = 0, x \in \Omega \subset \mathbb{R}^d, t \in [0, T]$$

  where $u(t, x)$ denotes the latent solution, $\mathcal{N}(\cdot;\ \lambda)$ is a nonlinear operator parametrized by λ.

- The above setup covers a wide range of PDEs in math and physics, including conservation laws, diffusion, reac-diff. E.g. Burger's equation:

$$\mathcal{N}(u;\ \lambda) = \lambda_1 u \cdot u_x - \lambda_2 u_{xx}, \lambda = (\lambda_1, \lambda_2)$$

- Given λ what is $u(t, x)$ (Data-driven solutions of PDEs)
- Find λ that best describes observations $u(t_i, x_j)$ (Data-driven discovery of PDEs)

Solving PDEs with Neural Network

Physics Informed Neural Network

- Rewrite the PDE as $\mathrm{f}(\mathrm{u};\mathrm{t},\mathrm{x})=0$

$$\mathrm{f}(\mathrm{u};\mathrm{t},\mathrm{x})=\mathrm{u}_{\mathrm{t}}+\mathscr{N}(\mathrm{u};\ \lambda)$$

along with $\mathrm{u}=\mathrm{u}_\theta(\mathrm{t},\mathrm{x})$.

- The loss function of PINN parameterized by θ is given by $\mathrm{L}=\mathrm{L}_{\mathrm{u}}+\mathrm{L}_{\mathrm{f}}$ where

$$\mathrm{L}_{\mathrm{u}}=\frac{1}{\mathrm{N}_{\mathrm{u}}}\sum_{\mathrm{i}=1}^{\mathrm{N}_{\mathrm{u}}}|\mathrm{u}(\mathrm{t}_{\mathrm{u}}^{\mathrm{i}},\mathrm{x}_{\mathrm{u}}^{\mathrm{i}})-\mathrm{u}^{\mathrm{i}}|^2,\mathrm{L}_{\mathrm{f}}=\frac{1}{\mathrm{N}_{\mathrm{f}}}\sum_{\mathrm{i}=1}^{\mathrm{N}_{\mathrm{f}}}|\mathrm{f}(\mathrm{t}_{\mathrm{f}}^{\mathrm{i}},\mathrm{x}_{\mathrm{f}}^{\mathrm{i}})|^2$$

Here $\{t_u^i,x_u^i,u^i\}_{i=1}^{N_u}$ denote the initial and boundary training data on $\mathrm{u}(\mathrm{t},\mathrm{x})$ and $\{t_f^i,x_f^i\}_{i=1}^{N_f}$ specify the collocation points for $\mathrm{f}(\mathrm{u};\mathrm{t},\mathrm{x})$.

Solving PDEs with Neural Network

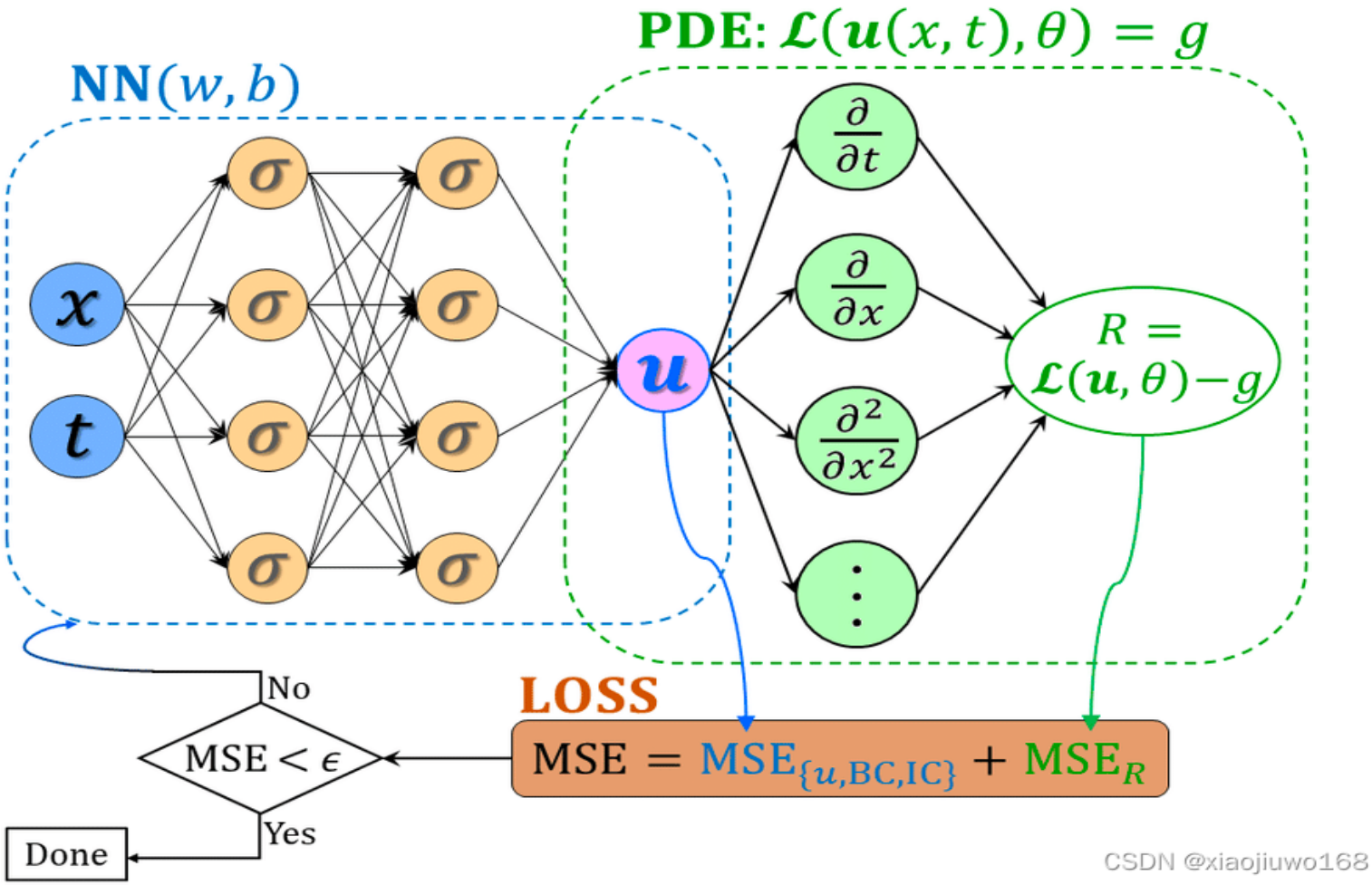

Figure 6: Continuous Time Models.

Solving PDEs with Neural Network

Physics Informed Neural Network

Schrödinger equation:

$$\begin{cases} f = ih_t + 0.5h_{xx} + |h|^2 h = 0, x \in [-5, 5], t \in [0, \pi/2] \\ h(0, x) = 2\mathrm{sech}(x), \\ h(t, -5) = h(t, 5), \\ h_x(t, -5) = h_x(t, 5) \end{cases}$$

Total loss is given by $L = L_0 + L_b + L_f = L_u + L_f$ where

$$L_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} |h(0, x_0^i) - h_0^i|^2, L_f = \frac{1}{N_f} \sum_{i=1}^{N_f} |f(t_f^i, x_f^i)|^2$$

$$L_b = \frac{1}{N_b} \sum_{i=1}^{N_b} \left(|h^i(t_b^i, -5) - h^i(t_b^i, 5)|^2 + |h_x^i(t_b^i, -5) - h_x^i(t_b^i, 5)|^2 \right)$$

Solving PDEs with Neural Network

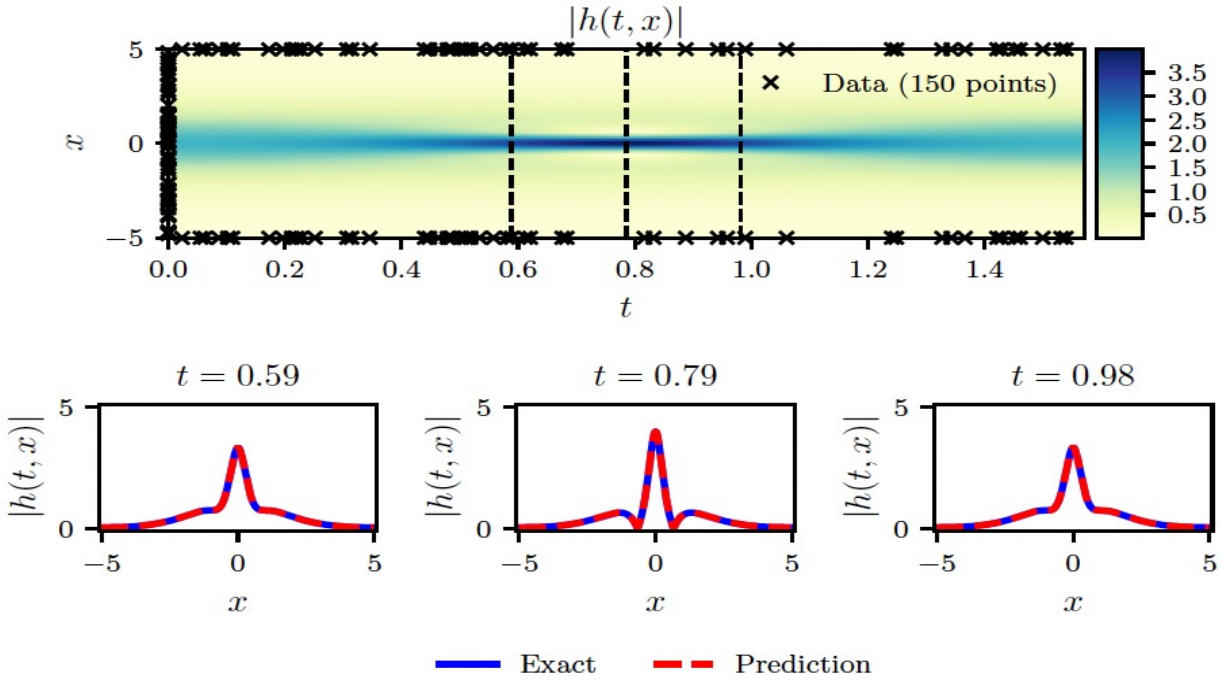

Figure 1: *Shrödinger equation: Top:* Predicted solution $|h(t,x)|$ along with the initial and boundary training data. In addition we are using 20,000 collocation points generated using a Latin Hypercube Sampling strategy. *Bottom:* Comparison of the predicted and exact solutions corresponding to the three temporal snapshots depicted by the dashed vertical lines in the top panel. The relative $\mathbb{L}_2$ error for this case is $1.97 \cdot 10^{-3}$.

Figure 7: Schrödinger equation: $\mathrm{N}_0 + \mathrm{N}_\mathrm{b} = 150, \mathrm{N}_\mathrm{f} = 20000$.

Solving PDEs with Neural Network

Physics Informed Neural Network

- $\mathrm{h}(\mathrm{t},\mathrm{x}) = \mathrm{u}(\mathrm{t},\mathrm{x}) + \mathrm{iv}(\mathrm{t},\mathrm{x})$ using a 5-layer deep neural network with 100 neurons per layer.
- The choice is purely empricial (no theoretical basis).
- Fine-tune the design of the DNN.
- Potential issues: Continuous time models require a large number of collocation points through the domain.

Solving PDEs with Neural Network

Examples of Physics Informed Neural Network

Solve the following 2-D problem in $\Omega = [0,1] \times [0,1]$:

$$\begin{cases} u_{xx} - u_{yyyy} = (2 - x^2)\exp(-y) \text{ in } \Omega, \\ u_{yy}(x,0) = x^2, \\ u_{yy}(x,1) = x^2/e, \\ u(x,0) = x^2, \\ u(x,1) = x^2/e, \\ u(0,y) = 0, \\ u(1,y) = \exp(-y) \text{ on } \partial\Omega. \end{cases}$$

where the true solution is

$$u(x,y) = x^2\exp(-y) \text{ in } \Omega.$$

Solving PDEs with Neural Network

Examples of Physics Informed Neural Network

Define following loss functions:

$$\begin{cases} L_1 = \frac{1}{N_1}\sum_{(x_i,y_i)\in\Omega}(u_{xx}(x_i,y_i;\theta) - u_{yyyy}(x_i,y_i;\theta) - (2-x_i^2)\exp(-y_i))^2, \\ L_2 = \frac{1}{N_2}\sum_{(x_i,y_i)\in[0,1]\times\{0\}}(u_{yy}(x_i,y_i;\theta) - x_i^2)^2, \\ L_3 = \frac{1}{N_3}\sum_{(x_i,y_i)\in[0,1]\times\{1\}}(u_{yy}(x_i,y_i;\theta) - x_i^2/e)^2, \\ L_4 = \frac{1}{N_4}\sum_{(x_i,y_i)\in[0,1]\times\{0\}}(u(x_i,y_i;\theta) - x_i^2)^2, \\ L_5 = \frac{1}{N_5}\sum_{(x_i,y_i)\in[0,1]\times\{1\}}(u(x_i,y_i;\theta) - x_i^2/e)^2, \\ L_6 = \frac{1}{N_6}\sum_{(x_i,y_i)\in\{0\}\times[0,1]}u(x_i,y_i;\theta)^2, \\ L_7 = \frac{1}{N_7}\sum_{(x_i,y_i)\in\{1\}\times[0,1]}(u(x_i,y_i;\theta) - \exp(-y_i))^2, \end{cases}$$

and

$$L = \omega_1 L_1 + \omega_2 L_2 + \omega_3 L_3 + \omega_4 L_4 + \omega_5 L_5 + \omega_6 L_6 + \omega_7 L_7$$

Solving PDEs with Neural Network

Conclusion

Advantages:

- PINN, a new class of universal function approximators that are capable of encoding any underlying physical laws.

Questions:

- How deep/wide should the neural network be? How much data is really needed?
- Does the network suffer from vanishing gradients for deeper architectures and higher order differential operators? Could this be mitigated by using different activation functions?
- Can we improve on initializing the network weights or normalizing the data? Loss function weight choices?

Thank you!