A brief overview of Physics-Informed Neural Networks and some critical remarks

Presentation · December 2024

DOI: 10.13140/RG.2.2.20276.16002

1 author:

Chennakesava Kadapa
Edinburgh Napier University
71 PUBLICATIONS **727** CITATIONS

SEE PROFILE

A brief overview of Physics-Informed Neural Networks and some critical remarks

Dr Chennakesava Kadapa

School of Computing, Engineering and the Built Environment
Edinburgh Napier University, Edinburgh, UK.
Email: `c.kadapa@napier.ac.uk`

02-Dec-2024

Neural Networks

- A NN model is basically a functional relationship between the input $(\mathbf{x})$ and output $(\mathbf{y})$, i.e.,

$$\mathbf{y} = \sigma^{(3)}(\mathbf{w}^{(3)}(\mathbf{i}) + \mathbf{b}^{(3)})$$
$$\mathbf{i} = \sigma^{(2)}(\mathbf{w}^{(2)}(\mathbf{h}) + \mathbf{b}^{(2)})$$
$$\mathbf{h} = \sigma^{(1)}(\mathbf{w}^{(1)}(\mathbf{x}) + \mathbf{b}^{(1)})$$

- Once the weights $(\mathbf{w})$ and biases $(\mathbf{b})$ of the chosen NN model are determined, neural networks can predict the output $(\mathbf{y})$ for the given input $(\mathbf{x})$.
- The key step in Machine Learning is finding the values of these weights and biases, which is known popularly as *training* of the Neural Network.

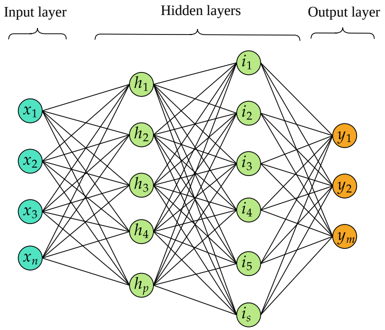


Figure: A Multi-input multi-output NN model.

Training of Neural Networks

- During the training phase of NNs, weights and biases are determined by minimising the loss function,

$$L = \frac{1}{2} \parallel \mathbf{y} - \hat{\mathbf{y}} \parallel^2 = \mathbf{e}^T \mathbf{e}. \tag{1}$$

- The error is defined as the difference between the predicted output $(\mathbf{y})$ and the actual output $(\hat{\mathbf{y}})$,

$$\mathbf{e} = \mathbf{y} - \hat{\mathbf{y}}$$

- Starting from an initial guess, the parameters are calculated using the backpropagation algorithm [1] in which the parameters are updated iteratively until the chosen convergence criterion is satisfied.

PINNs

- In PINNs, in addition to the loss function (1), loss functions based on the residual of the governing differential equation(s) as well as the residuals of initial and boundary conditions are considered [2].
- For example, for the Poisson equation for which the governing equation is,

$$\frac{\partial^2 u}{\partial x^2} = f,$$

the total loss function for is

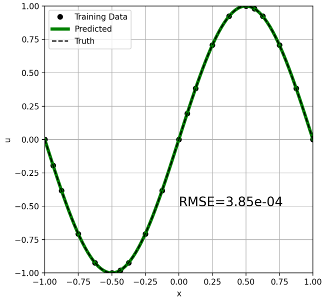
$$L = w_{\text{data}} L_{\text{data}} + w_{\text{PDE}} L_{\text{PDE}} \tag{2}$$

where w_{data} and w_{PDE} are the weights, and L_{data} and L_{PDE} are the loss functions.

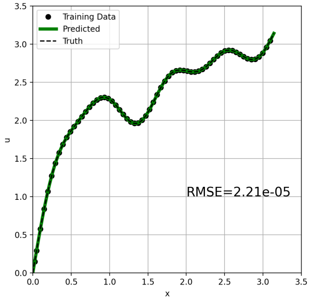
$$L_{\text{data}} = \frac{1}{2} \sum_{i=1}^{N_u} \|u(x_i) - \bar{u}(x_i)\|^2, \qquad L_{\text{PDE}} = \frac{1}{2} \sum_{j=1}^{N_c} \left\| \frac{\partial^2 u}{\partial x^2}(x_j) - f(x_j) \right\|^2 \tag{3}$$

where N_u and N_c are the number of data and collocation points.

PINNs - Examples with DeepXDE library



(a)

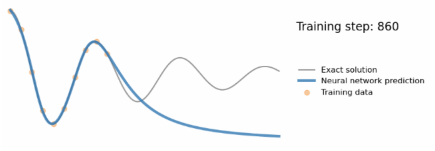


(b)

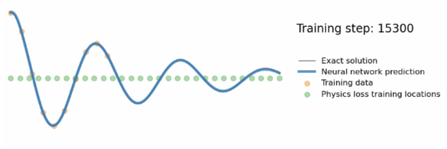
Figure: Predicted solutions for the Poisson equation with PINNs using the DeepXDE library [3]. (a) $y = \sin(\pi\, x)$ and (b) $y = x + \sin(x) + \sin(2x)/2 + \sin(3x)/3 + \sin(4x)/4 + \sin(8x)/8$ with hard BCs.

PINNs - Spring-mass-damper example [4]

- PINNs yield better results outside the domain of training data when compared with the naive NN model.



(a) Naive NN model



(b) PINNs model

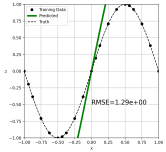
Figure: PINNs example from Ben Moseley's blog [4].

- But, look at the training steps needed for the PINN model.

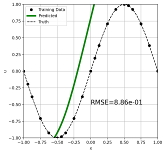
PINNs - weights

- If only boundary points are selected for the data points, the weights should be one. $w_{\text{data}} = w_{\text{PDE}} = 1$. This follow froms the point collocation method, see slide XYZ.
- If the data points include points inside the domain as well, then the value of weights should be adjusted.
  - For $N_u \ll N_c$,
    $$w_{\text{data}} \approx w_{\text{PDE}}$$
    .
  - For $N_u < N_c$,
    $$w_{\text{data}} \gg w_{\text{PDE}}$$
    .
  - For $N_u \approx N_c$,
    $$w_{\text{data}} \ggg w_{\text{PDE}}$$
- If $N_u \approx N_c$, there is no benefit of L_{PDE}. In fact, L_{PDE} degrades the performance. We need a deeper NN (excessive number of neurons).

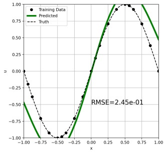
PINNs versus pure data-driven models - Example 1



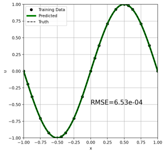
(a) 2 neurons



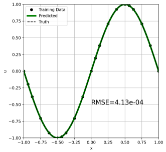
(b) 3 neurons



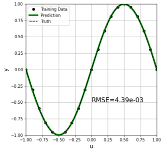
(c) 4 neurons



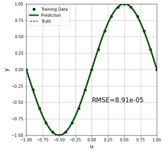
(d) 5 neurons



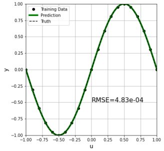
(e) 6 neurons



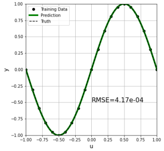
(f) 2 neurons



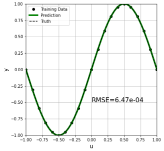
(g) 3 neurons



(h) 4 neurons



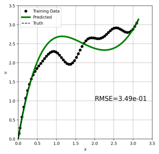
(i) 5 neurons



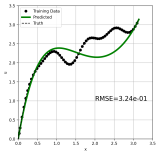
(j) 6 neurons

Figure: (Top row) With PINNs using the DeepXDE library and (Bottom row) With naive NN models using the Levenberg-Marquardt method.

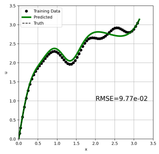
PINNs versus pure data-driven models - Example 2



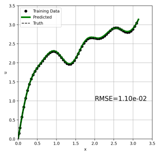
(a) 1 layer, 10 neurons



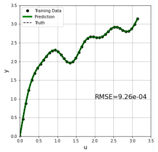
(b) 1 layer, 20 neurons



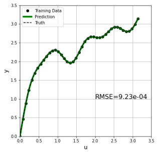
(c) 2 layers, 10 neurons



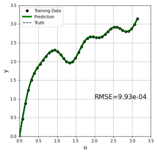
(d) 2 layers, 20 neurons



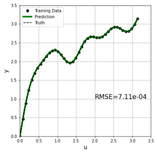
(e) 1 layer, 10 neurons



(f) 1 layer, 20 neurons



(g) 1 layer, 30 neurons



(h) 1 layer, 40 neurons

Figure: (Top row) With PINNs using the DeepXDE library and (Bottom row) With naive NN models using the Levenberg-Marquardt method.

PINNs versus pure data-driven models - Observations

- Adding PDE residuals as constraints seem to lower the accuracy.
- Pure data-driven NN models are better at capturing the input-output relations, as seen with the two examples.
- PINNs seem to struggle with increased nonlinearity. With PINNs, we need at least two layers and 20 neurons in each layer for example 2, while a naive NN model with 10 neurons accurately captures the response.
- Almost all the examples in the DeepXDE library, also many papers on PINNs, use excessive number of layers and neurons in each layer. Comprehensive studies on the performance with different neurons and layers are rare in the literature. So, this poor performance behaviour of PINNs is rarely discussed.

Collocation methods

- To understand PINNs, it is worth revisiting the point collocation method or collocation least-squares method [5, 6].
- Using the point collocation method, we can see why $w_{\text{data}} = w_{\text{PDE}} = 1$.

For the PDE

$$\mathcal{F}(x, u, u_x, u_{xx}, \ldots) = 0, \tag{4}$$

with the boundary conditions $u = \bar{u}$, we approximate the solution $u(x)$, e.g., $u(x) = c_0 + c_1\, x + \ldots + c_n\, x^n$, and find the coefficients by ensuring to satisfy

- the boundary conditions at the specified points $i = 1, 2, \ldots, N_b$, and
- the governing differential equation at the chosen collocation points, $x_j = 1, 2, 3, \ldots, N_c$.

Point collocation method - Example

$$\frac{\partial^2 u}{\partial x^2} = 2\,x \tag{5}$$
$$u(x=0) = 0 \tag{6}$$
$$u(x=1) = 1 \tag{7}$$
$$u_{exact}(x) = \frac{2}{3}\,x + \frac{1}{3}\,x^3 \tag{8}$$

Approximate function:

$$u(x) = c_0 + c_1\,x + c_2\,x^2 + c_3\,x^3 \tag{9}$$

At BCs:

$$u(x=0) = 0, \rightarrow c_0 = 0 \tag{10}$$
$$u(x=1) = 1, \rightarrow c_0 + c_1 + c_2 + c_3 = 1 \tag{11}$$

Second derivatie: $\frac{\partial^2 u}{\partial x^2} = 2\,c_2 + 6\,c_3\,x$

At collocation points, $x = 0.2, 0.4, 0.6, 0.8$:

$$2\,c_2 + 1.2\,c_3 = 0.4 \tag{12}$$
$$2\,c_2 + 2.4\,c_3 = 0.8 \tag{13}$$
$$2\,c_2 + 3.6\,c_3 = 1.2 \tag{14}$$
$$2\,c_2 + 4.8\,c_3 = 1.6 \tag{15}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 0 & 0 & 2 & 1.2 \\ 0 & 0 & 2 & 2.4 \\ 0 & 0 & 2 & 3.6 \\ 0 & 0 & 2 & 4.8 \end{bmatrix} \begin{Bmatrix} c_0 \\ c_1 \\ c_2 \\ c_3 \end{Bmatrix} = \begin{Bmatrix} 0.0 \\ 1.0 \\ 0.4 \\ 0.8 \\ 1.2 \\ 1.6 \end{Bmatrix}$$

Solution: $c_0, c_1 = 2/3, c_2 = 0, c_3 = 1/3.$

Point collocation method - Example (cont'd)

In the example problem for the point collocation method,

$$L_{\text{data}} = \frac{1}{2}\sum_{i=1}^{N_u=2} \|u(x_i) - \bar{u}_{x_i}\|^2, \qquad L_{\text{PDE}} = \frac{1}{2}\sum_{j=1}^{N_c=4} \left\|\frac{\partial^2 u}{\partial x^2}(x_i) - f_{x_i}\right\|^2 \tag{16}$$

Which can be solved for the coefficients $\mathbf{c}$ as

$$\left[w_{\text{data}}\,\mathbf{J}_1^T\,\mathbf{J}_1 + w_{\text{PDE}}\,\mathbf{J}_2^T\,\mathbf{J}_2\right]\,\mathbf{c} = \left[w_{\text{data}}\,\mathbf{J}_1^T\,\mathbf{f}_1 + w_{\text{PDE}}\,\mathbf{J}_2^T\,\mathbf{f}_2\right] \tag{17}$$

$$\mathbf{J}_1 = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \end{bmatrix}, \mathbf{J}_2 = \begin{bmatrix} 0 & 0 & 2 & 1.2 \\ 0 & 0 & 2 & 2.4 \\ 0 & 0 & 2 & 3.6 \\ 0 & 0 & 2 & 4.8 \end{bmatrix}, \mathbf{f}_1 = \begin{Bmatrix} 0 \\ 1 \end{Bmatrix}, \mathbf{f}_2 = \begin{Bmatrix} 0.4 \\ 0.8 \\ 1.2 \\ 1.6 \end{Bmatrix} \tag{18}$$

It is apparent that $w_{\text{data}} = w_{\text{PDE}} = 1$. This will change if we add data points in the domain.

Some observations

- For the linear PDEs, the solution using the point collocation method with polynomials can be calculated in a single step. However, since NN models are inherently nonlinear, we need iterative methods even for linear PDEs.
- Point collocation method results in least-square problems with poorly conditioned matrix systems when compared with conventional numerical methods (FEM, FDM, FVM). They also need higher-order derivatives.
- This is why point collocation methods didnot become popular when compared with FEM, FDM and FVM. This is not going to change with PINNs now.
- Due to the memory issues in storing the matrices, PINNs are solved without the matrices using the same backporpagation algorithms in Machine Learning libraries. Such iterative algorithms, e.g., steepest descent or gradient descent, are quite poor for solving the resulting problems due to PINNs. They are also highly sensitive to initial guesses.
- It is not a surprise that PINNs know *physics*, because PINNs actually solve the governing PDEs/ODEs, just like any other numerical method.

PINNs - Advantages

- The NN model can be *trained* with only boundary data and collocation points in the domain.
- No need for the simulation data, if we don't include data points within the domain.
- Adaptation of the existing framework for NN models.
- PINNs are applicable for any problem.

PINNs - Disadvantages

- Since NN models are inherently nonlinear - weights and biases from one layer depend on those from the other layers - we must use iterative methods to solve even linear PDEs/ODEs. Becomes more challenging for nonlinear problems.
- Need to create a different NN model for different PDE. Also, need to create a different NN model for different BCs and material properties. Exactly the same as solving using any other numerical method.
- If we want to include data points inside the domain as well, then we need to get it either from experiments or by running simulations using conventional solvers based on FDM, FEM, FVM etc. Then, there is no point adding additional loss functions of PDE residuals.
- If we have the simulation data within the domain, then there is no need for PINNs. Pure data-driven NN models (use only L_{data}) are better, as shown with the examples. Remember, $w_{\mathrm{data}} \ggg w_{\mathrm{PDE}}$ for $N_u \approx N_c$.

What about PINNs as surrogate models?

- Surrogate models with PINNs make zero sense since we actually solve the PDE using the collocation methods in PINNs.
- Since PINNs solve the PDE solution, using PINNs as surrogate models is the same as using the solution obtained with any other numerical method as surrogates. In this case, this is just like probing the solution in the post-processing step.
- If you are after surrogate models for 1D problems and 2D problems on rectangular domains, then spline or RBF fitting serves the purpose.
- For 2D problems with complex geometries or for 3D problems, pure data-driven NN methods could be better than PINNs if you have, or can get, the simulation data. *It is easier to fit the model than solve the PDE using point collocation method*.
- For parametrised PDEs, we still have to solve the PDEs for all different parameters. This is not better than solving PDEs using other numerical methods. It is just that we have the solution in one place, but getting to that solution would be quite difficult and resource intensive.

What about PINNs for inverse problems?

- For the inverse problems, we need the physics (the governing PDEs) anyway. So, PINNs are no special when compared to the standard methods used for inverse problems.
- For the ODEs, system identification methods perform much better [7].
- The advantages of PINNs for inverse problems will be apparent only if proper comparisons against conventional methods for inverse problems are made.
- Like, how are PINNs better in terms of cost? What inverse problems do PINNs solve that cannot be solved by the conventional methods? We need such comparative studies, but all we see are the methods applied to some problems without comparisons.

References I

M. T. Hagan, H. B. Demuth, M. H. Beale, and O. De Jesús.
Neural Network Design.
Second edition, 2014.

M. Raissi, P. Perdikaris, and G. E. Karniadakis.
Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.
Journal of Computational Physics, 378:686–707, 2019.

L. Lu, X. Meng, Z. Mao, and G. E. Karniadakis.
DeepXDE: A deep learning library for solving differential equations.
SIAM Review, 63(1):208–228, 2021.

B. Moseley.
So, what is a physics-informed neural network?
Website, August 2021.

References II

X. Zhang, X. Liu, K. Song, and M. Lu.
Least-squares collocation meshless method.
International Journal of Numerical Methods in Engineering, 51:1089–1100, 2001.

P. Bochev and M. D. Gunzburger.
Least-Squares Finite Element Methods.
Springer, New York, USA, 2009.

C. Kadapa.
Identification of nonlinear dynamical systems from time-series data.
2021.