Convergence analysis of Physics-Informed Neural Networks for solving inverse partial differential equations

Abhilash Neog, Amartya Dutta, Medha Sawhney

1 Introduction

Physics-informed neural networks (PINNs) are a class of machine learning models that combine neural networks with knowledge of the underlying physics governing a system. These models are particularly useful in situations where data is scarce or expensive to obtain, but there exists a known set of physical laws or equations that describe the behavior of the system.

The key idea behind PINNs is to embed the governing physical equations into the neural network architecture during the training process. This is achieved by incorporating the partial differential equations (PDEs) that represent the physics of the system as part of the loss function during the training of the neural network. In other words, the neural network is not only learning from the available data but is also constrained to satisfy the fundamental laws of physics.

Partial differential equations describe the relationship between a function and its partial derivatives with respect to one or more independent variables. They are commonly used to model physical phenomena in various fields, including physics, engineering, and finance.

A general form of a PDE can be written as:

$$F(u, \frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial y^2}, \ldots) = 0$$

Here, u is the unknown function, and $\frac{\partial u}{\partial x}, \frac{\partial u}{\partial y}, \frac{\partial^2 u}{\partial x^2}, \frac{\partial^2 u}{\partial y^2}, \ldots$ are its partial derivatives with respect to one or more independent variables (e.g., x and y).

In the context of PINNs, the idea is to incorporate the PDEs directly into the training process of a neural network. The loss function for training the neural network is composed of two main components:

1. **Data-Driven Loss**: This component ensures that the neural network fits the available data.

2. **Physics-Informed Loss**: This component enforces the neural network to satisfy the governing PDEs by penalizing deviations from the equations.

The formulation of the combined loss function of a Physics-Informed Neural Network (PINN) is given by:

$$\mathcal{L}_{total} = \mathcal{L}_{data} + \cdot\mathcal{L}_{physics} \tag{1}$$

The data-driven loss term ($\mathcal{L}_{data}$) is defined as:

$$\mathcal{L}_{data} = \sum_{i=1}^{N_{data}} \left(\hat{u}(x_i, t_i) - u_{observed}(x_i, t_i)\right)^2 \tag{2}$$

And the physics-informed loss term ($\mathcal{L}_{physics}$) is defined as:

$$\mathcal{L}_{physics} = \frac{1}{N_{physics}} \sum_{i=1}^{N_{physics}} (PDE_i)^2 \tag{3}$$

Here, $\hat{u}(x_i, t_i)$ is the predicted solution, $u_{observed}(x_i, t_i)$ is the observed solution, PDE_i represents the residual of the i-th partial differential equation (PDE) at a specific location and time, and λ is a hyperparameter controlling the weighting between the data-driven and physics-informed terms.

The combined loss function is minimized during training, leading to a neural network that not only accurately represents the observed data but also adheres to the underlying physical principles described by the PDEs.

By integrating knowledge of physics into the training process, PINNs can be powerful tools for making predictions in scenarios where limited data is available, and the underlying physical laws are well-defined. This approach has applications in fluid dynamics, heat transfer, structural mechanics, and other fields where PDEs govern the system behavior.

Related to the forward problem is the inverse problem. Solving inverse problems involving partial differential equations (PDEs) is a challenging and important task in various scientific and engineering applications. Inverse problems in the context of PDEs generally involve estimating the input or PDE parameters given observed output data. Physics-Informed Neural Networks (PINNs) have emerged as a promising approach for solving inverse problems related to PDEs. In the context of PINNs for inverse problems, the neural network is trained not only on the available data but also on the governing PDEs that describe the physical system. This allows the network to learn the PDE parameters while simultaneously fitting the observed data.

2 Problem Statement

The inverse PDE problem, despite being an important and impactful problem suffers from certain limitations. The primary issue is the difficulty in converging the physics-informed loss, which enforces the satisfaction of partial differential equations (PDEs). This is a common challenge when training Physics-Informed Neural Networks (PINNs) in general and not specific to the inverse problem.

It can be attributed to several factors, and addressing these issues may help improve the convergence of the physics-informed loss.

So, the goal of this project is to analyse the convergence of the PINN loss function under the inverse problem setting and consider techniques that can potentially improve the convergence problem.

In this project, we primarily focus on weighing the loss terms during training. The idea is that, since the data-driven loss converges easily and the PDE loss does not, we decrease the weight of the data-driven loss with training iterations while increasing the weight of the PDE loss term.

3 Methodology

3.1 Optimization technique

In this project, we have used the L-BFGS (Limited-memory BFGS) optimization algorithm, which is a variant of BFGS (Broyden-Fletcher-Goldfarb-Shanno) [1]. Both of them belongs to the family of quasi-Newton methods.

BFGS is an iterative optimization algorithm designed for unconstrained optimization problems. BFGS maintains an estimate of the inverse Hessian matrix, which is an approximation to the second-order derivatives of the objective function. At each iteration, BFGS updates the inverse Hessian approximation based on the gradients of the objective function and the changes in the parameter space. It terminates when a convergence criterion is met, such as reaching a specified tolerance in the gradient or achieving a specified tolerance in the change of the objective function.

L-BFGS (or LFGS) is a variant of BFGS designed for large-scale optimization problems. It is particularly useful when dealing with high-dimensional parameter spaces. L-BFGS stores a limited memory of the past iterations, making it more memory-efficient compared to the full BFGS. This limited-memory approach is especially beneficial when dealing with large datasets or high-dimensional problems. Moreover, instead of storing the full inverse Hessian matrix, L-BFGS maintains a limited-memory approximation, which significantly reduces the computational cost. The convergence criteria is same as that of BFGS.

3.2 Training Data

The data used for training the PINNs is obtained from the data published in the original PINN work [2]. In that work, the authors generate the PDE observed data synthetically. They employ numerical simulations to obtain the training data for the PINN.

The authors use standard numerical methods, such as finite difference or finite element methods, to solve the given partial differential equations (PDEs) numerically. These simulations provide a set of solutions at various points in the spatio-temporal domain.

The continuous PDEs are then discretized over a spatial and temporal grid. The numerical solver is then used to compute the values of the solution at discrete points within this grid. The values of the solution, as well as the corresponding spatial and temporal coordinates, obtained from the numerical simulations are used as the observed data for training the PINN. This dataset includes pairs of input points (x, t) and the corresponding PDE solution $u(x, t)$.

The PINN is trained using this dataset to learn the underlying physics and approximate the solution to the PDE. The network is trained to minimize the discrepancy between the predicted solution and the actual solution at the provided data points.

It's important to note that the synthetic nature of the dataset allows the PINN to learn the physics encoded in the PDE without requiring a large amount of real-world data. This is a key advantage of PINNs, especially in scenarios where obtaining real-world data might be expensive, impractical, or challenging.

3.3 Model training

3.3.1 Model architecture

PINNs are implemented as simple multi-layer neural networks. The input Layer takes input features, (in case of Bugers equation, spatial and temporal coordinates). The hidden layers comprise a series of densely connected layers with activation functions, allowing the network to capture complex patterns. They are followed by the output Layer, which produces predictions for the solution variables.

3.3.2 Loss function

Loss comprise of two losses (as explained above). Data-Driven Loss measures the discrepancy between the predicted solution and the observed data. Physics-Informed Loss enforces the PDE residuals to be close to zero, ensuring that the neural network respects the underlying physics.

3.3.3 Model Parameters and hyper-parameters

The parameters and hyper-parameters employed during training the PINNs are as follows:

- Number of layers = 10
- Number of neurons in the hidden layers = 20
- Termination tolerance on first order optimality (gradient tolerance for termination) = 1e-5
- Stepsize = 1.0
- Line search function = Strong wolfe

4 Experiments and Analysis

In the experimentations, following equations are considered,

Burgers Equation

$$u_t + uu_x - (0.01/\pi)u_{xx} = 0, x \in [1, 1], t \in [0, 1]$$

$$u(0, x) = -sin(\pi x)$$

$$u(t, -1) = u(t, 1) = 0$$

So, the PDE, $f(t, x)$ is defined as,

$$f := u_t + uu_x - (0.01/\pi)u_{xx}$$

where, $u(t, x)$ is approximated by the deep neural network

$l1, l2$ in the experiments refer to 1 and $0.01/\pi$ respectively

Allen-Cahn Equation

$$u_t - 0.0001u_{xx} + 5u^3 - 5u = 0, x \in [-1, 1], t \in [0, 1]$$

$$u(0, x) = x^2 cos(\pi x)$$

$$u(t, -1) = u(t, 1)$$

$$u_x(t, -1) = u_x(t, 1)$$

$l1, l2$ in the experiments refer to 0.0001 and 5 respectively

4.1 Without weighted loss

In this experiment, we consider the Burgers equation. The model almost converges after 10000 iterations using LFGS, i.e. fits the function u(x,t) almost perfectly. The loss is quite low. However, errors in parameter estimation is quite high, indicating that the physics-guided loss function has not yet converged

10000 Max Iterations
Error u: 9.207878e-02
Error l1: 25.01350%
Error l2: 23.93044%

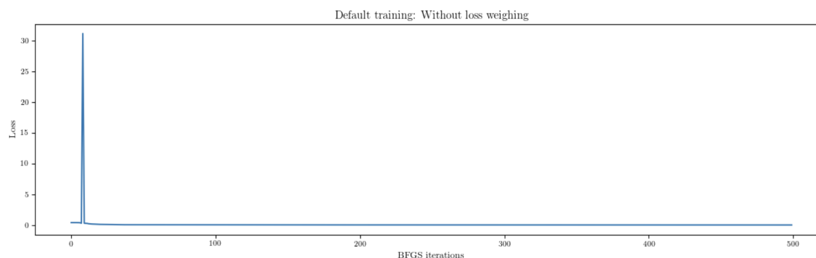

Figure 1: Loss curve with LFGS and no weighing (max 10k iterations)

Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Identified PDE (clean data)	$u_t + 0.74986uu_x - 0.0039448u_{xx} = 0$

Figure 2: Predicted and ground-truth parameters with max 10k iterations

50000 Max iterations
Error on u: 5.206837e-03
Error l1: 0.41518%
Error l2: 4.47659%

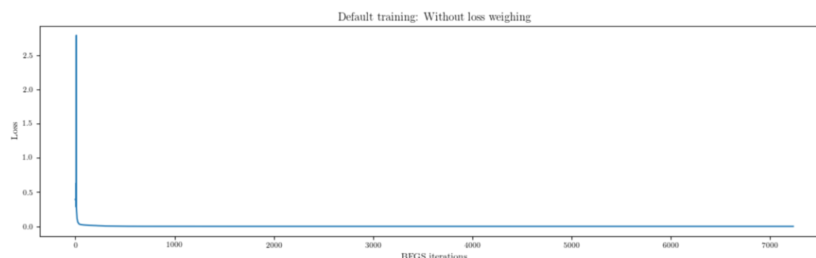

Figure 3: Loss curve with LFGS and no weighing (max 50k iterations)

Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Identified PDE (clean data)	$u_t + 0.99974uu_x - 0.0031788u_{xx} = 0$

Figure 4: Predicted and ground-truth parameters with max 50k iterations

From the above experiment, we can see that, while the neural network seems to fit the function u almost perfectly after around just 10000 iterations, it is still far away from estimating the actual parameters of the PDE. But, as we increase the number of iterations from 10000 to 50000, both the losses seems to be converging with the model predicting the PDE parameters relatively more accurately. This confirms the hypothesis that both the loss functions do

not converge simultaneously, and the loss on the u is not sufficient to understand the convergence of PINNs on the inverse problem.
So, there are two questions here,

1. Number of iterations had to be increased from 10000 to 50000 to actually reach the final convergence. How do we know what is an optimal number of iterations here?

2. Since, one loss converge quickly, can we perform adaptive weighing of losses? Or is a sequential training better?

4.2 With uniform weighted loss

In this experiment, we perform uniform weighted loss on the burgers equation i.e. we increase and decrease the weight of both the loss term uniformly. Following figure shows how the loss terms are varied with the iterations (in the figure, we have considered 8000 iterations to visualize). The yellow curve is the loss weight applied on the loss obtained fitting the function u and the blue line, which is increasing, is applied on the loss defined for the PDE parameters.

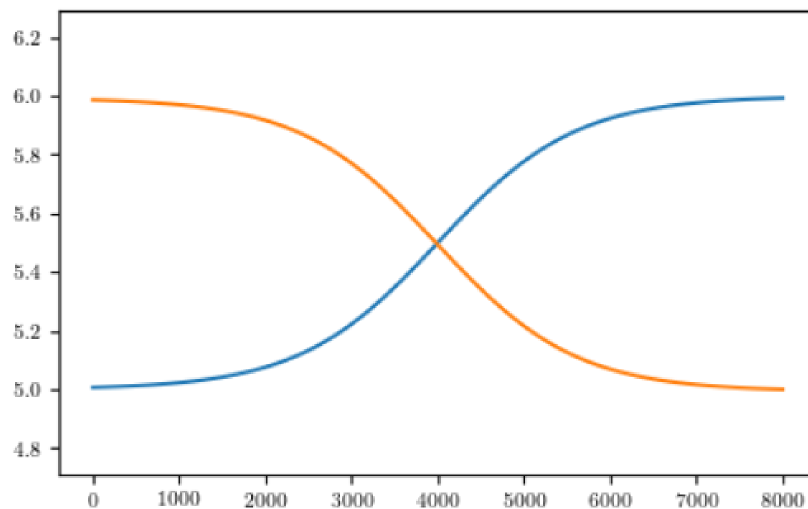

Figure 5: Changing weights w.r.t iterations.

10000 Max iterations
Error u: 5.688636e-03
Error l1: 0.39384%
Error l2: 4.36655%
50000 Max Iterations
Error u: 4.666111e-03

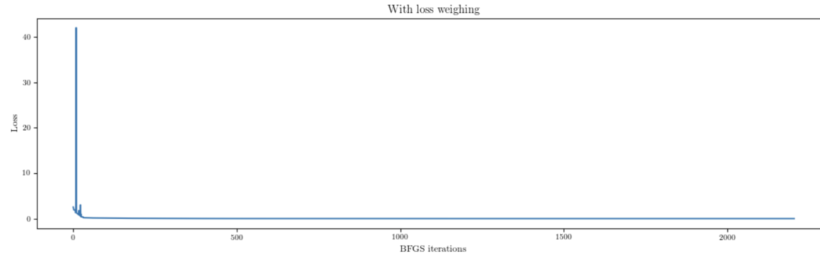

Figure 6: Loss curve with LFGS and loss weighing (max 10k iterations)

Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Identified PDE (clean data)	$u_t + 0.99606uu_x - 0.0033221u_{xx} = 0$

Figure 7: Predicted and ground-truth parameters with max 10k iterations

Error l1: 0.05592%
Error l2: 1.44720%

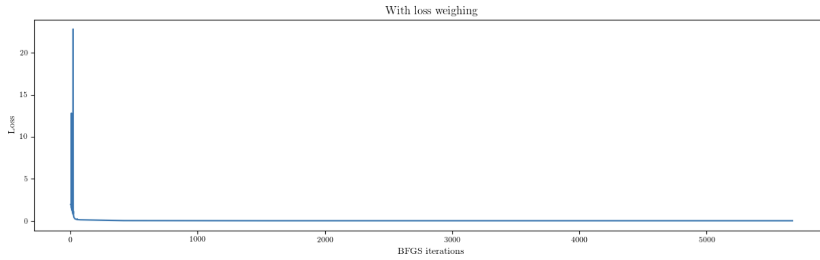

Figure 8: Loss curve with LFGS and loss weighing (max 50k iterations)

Correct PDE	$u_t + uu_x - 0.0031831u_{xx} = 0$
Identified PDE (clean data)	$u_t + 0.99944uu_x - 0.0032292u_{xx} = 0$

Figure 9: Predicted and ground-truth parameters with max 50k iterations

As seen above, with adequately weighing the loss terms, both loss functions converge with a significant reduction in the parameter estimation errors for 10k iterations in the weighted case compared to the unweighted case.

4.3 Results

The above experiments were conducted for a total of two equations (for max 10k iterations) - Burgers equation and Allen-Cahn equation. Table 1 captures the respective performances

	u_{loss}		$l1_{loss}$		$l2_{loss}$	
	unweighted	weighted	unweighted	weighted	unweighted	weighted
Burger	9.207878e-02	**5.688636e-03**	25.01350%	**0.39384%**	23.93044%	**4.36655%**
Allen-Cahn	**2.610683e-01**	2.668656e-01	98.01003%	**2.00300%**	1006.07452%	**499.68213%**

Table 1: Losses for Different Equations with max lfgs iterations=10k

4.4 Robustness to noise

We add some gaussian noise to the actual data and train the model on noisy data to see if the model can learn under adversarial settings or not. We evaluate for the both the cases - weighted and unweighted. The following table records the results obtained on training the model for 10000 iterations (since with 50k iterations the model anyway converges).

	u_{loss}		$l1_{loss}$		$l2_{loss}$	
	unweighted	weighted	unweighted	weighted	unweighted	weighted
Burger	9.207878e-02	**5.688636e-03**	25.46611%	**0.20681%**	35.70578%	**0.42538%**
Allen-Cahn	**2.610683e-01**	2.668656e-01	**2.00830%**	2.28831%	**499.54843%**	499.70331%

Table 2: Losses for Different Equations trained on noisy data with max lfgs iteration set to 10k

A very interesting observation during PINN training - the loss on noisy data was significantly less compared to that of clean data. This raises multiple questions,

1. Do PINNs converge better on noisy data? and why

2. Another relevant question here - can slightly noisy data resemble the real-world better than the simulation data being used for training?

4.5 Sequential training

Does sequential training help? Is it better than performing loss weighing? Sequential training basically refers to training the model in 2 stages - in the first stage we try to train the model to fit the function u; in the second stage we train the model to reduce the physics loss, i.e.

1. Stage 1: Train on data driven loss

2. Stage 2: Train on physics loss

The hypothesis was that, joint training makes the model confused whether to fit the function u or try to get the PDE parameters right. However, as the experiment results on Burgers suggest, joint training is better than training sequentially.
For the sequential training experiment, instead of the LFGS we use an Adam Optimizer. The reason behind this is that, in this sequential case with LFGS, we observed that the algorithm was terminating even before 5000 max iterations were reached, hence, there was no difference in trying different higher max iterations, say 10000.

Iterations (Stage 1, Stage 2)	u_{loss}	$l1_{loss}$	$l2_{loss}$
5000, 5000	8.205882e-03	97.55043%	27.55819%
10000, 10000	7.492132e-03	97.85362%	24.36492%

Table 3: Sequential training results. Values shown are obtained after the 2 stage training

As can be seen from the table above, sequential training is significantly worse than loss weighing. Although it was not tested on multiple equations, the fact that we are getting poor results on a relatively simple equation suggests that sequential training is not an efficient solution.

5 Discussion and Future work

1. Weighing helps in better convergence. In both the cases, Burgers and Allen-Cahn, with weighing, the losses were significantly improved
2. This does not conclude that weighing is an optimal solution. Moreover, the weighing performed in our case was uniform, using a sigmoid-like function. This may not be the optimal case, unless the same technique is tested on all the standard benchmark PDEs.
   (a) The frequency at which weights are updated is currently set to 100 iterations. This may not be the optimal choice
   (b) Is uniformly changing the weights correct? Can we do a loss-based adaptive weighing? This would help converge better than independent weighing (current approach)
3. Joint training PINNs is better than sequential training, especially for the inverse PDE problem. While we didn't dive deep into the exact reason behind this, a simple intuition would be that neural network learning saturates onto the current task after certain number of training iterations and using that trained model to train on a different task does not help the model to really learn it. The term used to describe this phenomenon is known as, catastrophic forgetting.

References

[1] Broyden, C. G., Fletcher, R., Goldfarb, D., & Shanno, D. (1970). *Conjugate gradient methods for unconstrained minimization problems.* Mathematics of Computation, 24(111), 23-34.

[2] Raissi, Maziar, Paris Perdikaris, & George E. Karniadakis (2019). *Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations.*. Journal of Computational Physics 378 (2019): 686-707.