


# Using Scientific Machine Learning to solve Partial Differential Equations

Miles Cochran-Branson<sup>1</sup> 

<sup>1</sup>Department of Physics and Mathematics, Lawrence University, Appleton, WI.

**Keywords:** SciML, Scientific Computing, ML/AI, PINN, PDEs

## Abstract

Here is my abstract!

## Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Machine Learning</b>	<b>1</b>
2.1	Neural Networks . . . . .	1
2.2	Training Neural Networks . . . . .	1
2.3	The Universal Approximation Theorem . . . . .	1
2.4	Solving ODEs with Physics-Informed Neural Networks . . . . .	2
<b>3</b>	<b>Solving PDEs with <code>NeuralPDE.jl</code> using SciML</b>	<b>2</b>
<b>4</b>	<b>Conclusion</b>	<b>3</b>

## 1. Introduction

Machine learning and scientific computing are two prominent fields with applications in physics, mathematics, and computer science. In the past several years, potential has been seen in combining the fields of machine learning and scientific computing to overcome two potential problems: lack of data, and solving differential equations. [1]

## 2. Machine Learning

### 2.1. Neural Networks

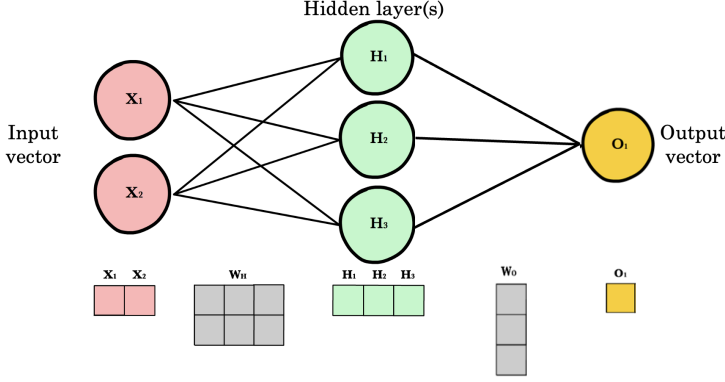
A neural network is simply a function.

### 2.2. Training Neural Networks

### 2.3. The Universal Approximation Theorem

Thing that makes everything work!

**Theorem 2.1.** Consider  $x \in \Omega$  and let  $u(x)$  be a regular function and  $N(x, w)$  be a neural network with weights  $w$ . Then we can find some arbitrarily small  $\epsilon \in \Omega$  such that



**Figure 1.** An example of a simple neural network with two inputs, one hidden layer with three nodes, and a single output. The network is defined by the weights connecting the layers which are represented as matrices. At each node, the weights are passed to a function called the activation function which scales the weights appropriately. In principle, the size of the network can change in every dimension. .

$$||N(x, w) - u(x)|| < \epsilon. \quad (2.1)$$

#### 2.4. Solving ODEs with Physics-Informed Neural Networks

### 3. Solving PDEs with NeuralPDE.jl using SciML

We now turn our attention to solving partial differential equations using techniques from scientific machine learning. We will use the package `NeuralPDE.jl` [2] to solve some examples.

Consider a partial differential equation given by

$$f(u(x); \lambda) = 0 \quad (3.1)$$

where  $x \in \Omega$  are the independent variables in the space  $\Omega$ ,  $u(x)$  is the solution,  $f$  is some non-linear function acting on  $u$  and  $\lambda$  are the parameters of the equation. To solve with a neural network  $N(x, w)$  where  $N$  is a neural network with weights  $w$ , we want to fulfill

$$f(N(x, w); \lambda) \approx 0. \quad (3.2)$$

The universal approximation theorem tells us that this should be possible. The error of the above equation is then given by

$$L(w) = \int_{\Omega} ||f(N(x, w); \lambda)|| dx. \quad (3.3)$$

We have suggestively named the error  $L$  as we can use this function as a loss function in training our neural network  $N$  by attempting to minimize  $L$ . Our computational problem then reduces to the problem of evaluating the integral in Eqn. 3.3. Notice that in practical problems, we apply boundary conditions  $b_i$  on  $\partial\Omega \in \Omega$ . In order to use these in our neural network, we add in these conditions to our loss function giving us

$$L(w) = \sum_i \int_{\Omega \setminus \partial\Omega} ||f_i(N(x, w); \lambda)|| dx + \sum_i \int_{\partial\Omega} ||b_i(N(x, w); \lambda)|| dx. \quad (3.4)$$

where we have generalized the problem to a system of coupled differential equations  $f_i$ .

There are several different methods for evaluating the above integral namely: grid approximation techniques, stochastic grid approximation techniques, and machine learning (quadrature) techniques. A simple grid approximation takes the space  $\Omega$  and divides it into units of volume  $\Delta V$  with a specific length. The PDE is evaluated at each of this points and scaled by the volume, thus our integral is computed via

$$\int_{\Omega} ||f(N(x, w); \lambda)|| dx = \sum_i \Delta V ||f(x_i)||. \quad (3.5)$$

This method has two main disadvantages: 1) as the dimension of the problem increases, the number of points we sample increases exponentially in order to maintain the same granularity and 2) no evaluation of the integral is done between grid points, thus information can be quickly lost. To solve the second problem, we can use stochastic methods of sampling—i.e., use Monte Carlo techniques to evaluate the integral. This can be described by

$$\int_{\Omega} ||f(N(x, w); \lambda)|| dx = \alpha \sum_i ||f(x_i)||. \quad (3.6)$$

for some scaler  $\alpha$ . As we simply want to minimize this integral, there is no need to specify  $\alpha$ . This problem still suffers from dimensionality exponentially increasing. Thus, we turn to a third method: quadrature training with a neural network. Several processes for specifying quadrature are described in [3]. These are typically of the form

$$\int_{\Omega} ||f(N(x, w); \lambda)|| dx = \sum_i \alpha_i ||f(x_i)|| dx. \quad (3.7)$$

In the implementation via `NeuralPDE.jl`, a neural network is solved via `Integrals.jl` which calls the `JuliaDifferential` equation solver [4] to find the correct sampling points  $x_i$  and weights  $\alpha_i$ .

## 4. Conclusion

**Acknowledgments.** Grateful for support of Pickett and Heaton + physics and mathematics EDU at LU! Huzzah!!

**Data Availability Statement.** A statement about how to access data, code and other materials allowing users to understand, verify and replicate findings — e.g. Replication data and code can be found in Harvard Dataverse: `\url{https://doi.org/link}`.

## References

- [1] G. E. Karniadakis, I. G. Kevrekidis, L. Lu, P. Perdikaris, S. Wang, and L. Yang, “Physics-informed machine learning,” *Nature Reviews Physics*, vol. 3, pp. 422–440, June 2021.
- [2] K. Zubov, Z. McCarthy, Y. Ma, F. Calisto, V. Pagliarino, S. Azeglio, L. Bottero, E. Luján, V. Sulzer, A. Bharambe, N. Vinchhi, K. Balakrishnan, D. Upadhyay, and C. Rackauckas, “NeuralPDE: Automating Physics-Informed Neural Networks (PINNs) with Error Approximations,” July 2021.
- [3] J. A. Rivera, J. M. Taylor, Á. J. Omella, and D. Pardo, “On quadrature rules for solving Partial Differential Equations using Neural Networks,” *Computer Methods in Applied Mechanics and Engineering*, vol. 393, p. 114710, Apr. 2022.
- [4] C. Rackauckas and Q. Nie, “DifferentialEquations.jl – A Performant and Feature-Rich Ecosystem for Solving Differential Equations in Julia,” *Journal of Open Research Software*, vol. 5, p. 15, May 2017.