ESKİŞEHİR TECHNICAL UNIVERSITY JOURNAL OF SCIENCE AND TECHNOLOGY
A- APPLIED SCIENCES AND ENGINEERING

Estuscience – Se, 2024, 25 [4] pp. 530-541, DOI: 10.18038/estubtda.1470050

RESEARCH ARTICLE

ADVANCED APPLICATIONS OF PHYSICS-INFORMED NEURAL NETWORKS (PINNS) IN R FOR SOLVING DIFFERENTIAL EQUATIONS

Melih AGRAZ [1, *]

[1] Department of Statisitcs, Giresun University, Giresun, Türkiye
melih.agraz@giresun.edu.tr, melihagraz@gmail.com- 0000-0002-6597-7627

Abstract

Deep learning, a powerful machine learning technique leveraging artificial neural networks, excels in identifying complex patterns and relationships within data. Among its innovations is the emergence of Physics-Informed Neural Networks (PINNs), which have revolutionized the field of applied mathematics by enabling the solution and discovery of differential equations through neural networks. PINNs address two key challenges: data-driven solutions, where the model approximates the hidden solutions of differential equations with fixed parameters, and data-driven discovery, where the network learns parameters that best describe observed data. This study explores the implementation of PINNs within the R programming environment to solve two differential equations: one with boundary conditions $y' - y = 0$ with y(0)=0 and y(e)=1 boundaries and the Burgers' Equation. The research utilizes R libraries, including reticulate for Python integration and torch for neural network operations, to demonstrate the versatility and efficacy of PINNs in addressing both data-centric solutions and parameter discovery. The results showcase the ability of PINNs to handle complex, high-dimensional problems, offering a promising alternative to traditional numerical methods for solving differential equations.

Keywords

Physics-Informed Neural Networks (PINNs), Differential Equations, R-programming language, Burgers' Equation

Time Scale of Article

Received :17 April 2024
Accepted : 25 November 2024
Online date :27 December 2024

1. INTRODUCTION

Differential equations, encompassing Ordinary Differential Equations (ODEs) and Partial Differential Equations (PDEs), serve as mathematical models for describing the dynamics of systems across various disciplines. Since Newton's *Principia*, these equations have been fundamental to understanding and explaining physical laws. The primary objective of differential equations is to derive solutions that adhere to governing mathematical expressions characterizing the phenomena under study. Consequently, devising efficient and accurate methods to solve these equations is critical for scientific advancement and engineering applications.

Deep Learning (DL), a branch of machine learning, employs artificial neural networks with multiple layers to tackle problems involving regression, pattern recognition, and classification. While traditionally DL has not been focused on solving differential equations, recent developments highlight its potential in this domain. The pioneering work by Lagaris et al. [1] laid the groundwork by employing artificial neural networks for boundary and initial value problems. Cheng et al. [2] extended this idea to Hamilton-Jacobi-Bellman equations. More recently, the advent of Physics-Informed Neural Networks (PINNs), introduced by Raissi et al. [3-5] under the mentorship of Karniadakis and colleagues, marked a significant breakthrough. PINNs utilize deep learning methodologies to address forward and inverse problems in PDEs, offering an innovative alternative to conventional solvers.

*Corresponding Author: melihagraz@gmail.com

Several R packages exist for solving differential equations. For instance:

- The **deSolve** package (Soetaert et al. [6]) addresses initial boundary problems for ODEs and PDEs.
- The **ReacTran** package (Soetaert and Meysman [7]) focuses on reactive transport equations in 1D, 2D, or 3D domains.
- The **rootSolve** package (Soetaert [8]) employs the Newton-Raphson method to determine roots of nonlinear and linear equations.

In contrast, Python provides robust support for PINNs through libraries like PyTorch (Paszke et al. [9]) and TensorFlow (Abadi et al. [10]). However, the absence of an equivalent R package for PINNs presents a gap in the R ecosystem. This study addresses this limitation by demonstrating the implementation of the PINNs approach in R, paving the way for the development of a dedicated R package.

In this study, we first introduce the methodology of PINNs and provide a comprehensive overview of their workings. We then present two examples to illustrate how PINNs can be applied in R. These examples cover the definition of loss functions and parameter prediction, highlighting the effectiveness and versatility of this approach.

2. MATERIAL and METHODS

2.1. Overview of Physics-informed Neural Networks (PINNs)

Physics-Informed Neural Networks (PINNs) were initially proposed by Karniadakis, Raissi, and colleagues [3-5] as a neural network-based approach for solving partial differential equations (PDEs). A significant advancement in PINNs was the incorporation of a residual network, which represents a major innovation. This network incorporates the governing physical equations, utilizes the output of the deep learning model, and calculates the residual values, as highlighted by Markidis [11].

An equation that illustrates the general form of the partial differential equation addressed by Physics-Informed Neural Networks (PINNs) is as follows:

$$\begin{aligned} u_t + N\,[u;\lambda] = 0, x \in \Omega, t \in [0,\Omega], t \in [0,T] \\ u(0,x) = h(x), x \in \Omega \\ u(t,x) = g(t,x), x \in \partial\Omega, t \in [0,T] \end{aligned} \tag{1}$$

In this framework, *u* (that is, *u(t, x)*) denotes the unknown solution subject to boundary conditions *g(t, x)* and initial conditions *h(x)*; essentially, *u* is the target variable of interest (e.g., representing a wave). The derivative u_t denotes the partial derivative of *u* with respect to time *t* over the interval [0, T], while *x* is an independent spatial variable within the domain Ω. In other words, *x* and *t* are the given inputs (e.g., spatial location *x* and time *t*), and *N[u; λ]* is a linear or nonlinear differential operator characterized by a set of PDE parameters λ.

In solving the differential equation, the function *u* is approximated using a fully connected deep neural network, where *(t, x)* serve as inputs and $u_{NN}(t,x)$ as outputs. A deep neural network comprises multiple hidden layers, each of which has inputs (X = [x_1 , x_2,..., x_i]) and outputs (Y = [y_1 , y_2, ..., y_i]).

To simplify the equation, the left-hand side ($u_t + N\,[u;\lambda]$) can be expressed as *f(t, x),* that is,

$$f(t,x) := u_t + N[u;\lambda] \tag{2}$$

The artificial neural network is constructed using hidden layers, where the inputs and outputs of the layers are transmitted throughout the network according to the formula:

$$\sigma(w_{i,j}x_i + b_i), \tag{3}$$

where b_j and $w_{i,j}$ represent biases and weights, respectively. σ(.) denotes the activation function, typically applied as a hyperbolic tangent activation function for each neuron except for the last layer, where no additional regularization is applied Cai et al. [12]. The parameters of the neural networks shared between u(t, x) and f(t, x) are learned by minimizing the loss function.

$$MSE = MSE_r + MSE_b + MSE_0, \tag{4}$$

where,

$$MSE_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} |u(t^i, x^i) - h^i|^2 , \tag{5}$$

and

$$MSE_b = \frac{1}{N_b} \sum_{i=1}^{N_b} |u(t^i, x^i) - g^i|^2 , \tag{6}$$

and

$$MSE_r = \frac{1}{N_r} \sum_{i=0}^{N_r} |u_t(t^i, x^i) + N_x u(t^i, x^i)|^2 . \tag{7}$$

Here in Equations 4-7, MSE_0, MSE_b , and MSE_r correspond to the losses associated with initial conditions, boundary conditions, and the penalization of residuals in the governing equations, respectively. To compute the residuals for MSE_r, it is necessary to obtain the derivatives of the outcomes—namely, u_t and $N_x u$ with respect to the inputs. These derivatives are calculated using automatic differentiation, as described by Baydin et al. [13]. The overall loss function is then optimized using an algorithm such as stochastic gradient descent (Ruder [31]) or the ADAM optimizer (Kingma and Ba [14]), among others.

An illustration of the PINNs approach is provided in Figure 1 below. As depicted in Figure 1, a fully-connected deep feed-forward neural network is utilized to approximate *u(t, x)*. This approximation is subsequently utilized to formulate the initial conditions loss MSE_0, the boundary conditions loss MSE_b, and residual loss MSE_r.

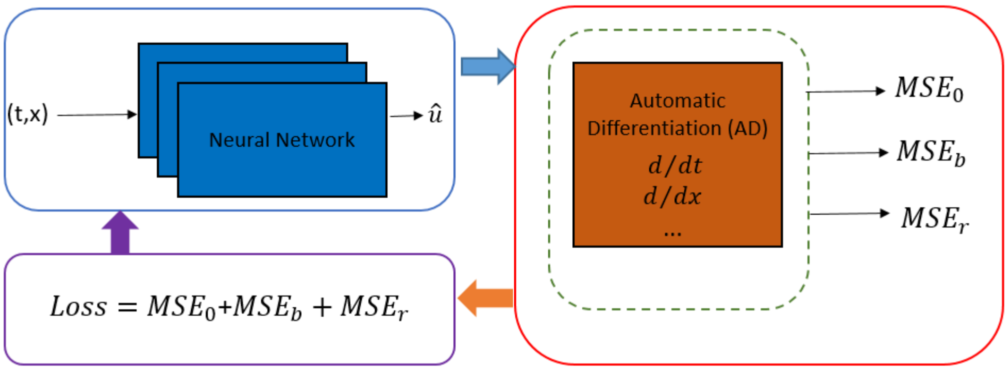


Figure 1. Physics-Informed Neural Networks (PINNs) Workflow [32].

Since their introduction, Physics-Informed Neural Networks (PINNs) contributed a significant impact on fluid mechanics and scientific computing, leading to notable advancements. Karniadakis and his collaborators have built upon these methodologies, resulting in various extensions of PINNs. These extensions include stochastic PINNs (Zhang et al. [15]), fractional PINNs (fPINNs) (Pang et al. [16]), conservative physics-informed neural networks (CPINNs) (Jagtap et al. [17]), parareal physics-informed neural networks (PPINNs) (Meng et al. [18]), extended physics-informed neural networks (XPINNs) (Jagtap and Karniadakis [19]), non-local PINNs (nPINNs) (Pang et al. [20]), PINNs with a variational formulation based on the Galerkin method (hp-VPINN) (Kharazmi et al. [21]), parallel PINNs (Shukla et al. [22]), Bayesian PINNs (Yang et al. [23]), and approaches for learning nonlinear operators via DeepONet (Lu et al. [24]). Agraz et al. [32] showed that simple differential equations can be effectively solved using a single multiplicative neuron.

PINNs can address two distinct problems: data-driven solutions for PDEs and data-driven discovery. In the first scenario, the model parameter λ remains constant, and PINNs approximate the hidden solution. In the latter scenario, PINNs are employed to identify the λ parameter that best characterizes the observed data [3, 25]. This article presents examples and definitions of both solution types, focusing on two different problems to illustrate the approach's efficacy. First, we begin by showcasing the solution to the differential equation $y' - y = 0$ with the initial conditions $y(0) = 1$ and $y(1) = e$. This is achieved through the utilization of the PINNs methodology, complemented by the incorporation of the reticulate package for data-centric solutions. Second, we tackle the one-dimensional Burgers' equation Basdevant et al. [26] using PINNs, employing the torch package Falbel and Luraschi [29] to exemplify a data-driven discovery scenario. The Burgers' equation is a fundamental partial differential equation that stems from the Navier-Stokes equations Raissi et al. [5]. Within this section, we embark on elucidating the foundational aspects of PINNs within the context of the R community. We commence this exploration with a clear-cut example centered around a simple differential equation. To facilitate a comprehensive understanding, the complete code for this illustrative instance is available in Supplementary A.1 on GitHub. Subsequently, we employ the torch package to tackle the one-dimensional Burgers' Equation as put forth by Basdevant et al. [26], leveraging the prowess of the PINNs approach to unveil insights driven by data. For interested readers, the comprehensive code pertaining to this particular example can be found in Supplementary A.2 on GitHub.

2. RESULTS

Data-Driven Solution of the $y' - y = 0$

To provide a straightforward introduction to the concept of PINNs, we offer an example that illustrates how to implement the PINNs approach for solving differential equations from the ground up using R

4.1.3. We begin by outlining a motivating example of a differential equation and its associated initial conditions, as defined in Equation8.

$$y' - y = 0, y(0) = 1, y(1) = e \tag{8}$$

We initiate by installing and loading the pertinent packages and creating a Python virtual environment named "r-reticulate." The reticulate package Ushey et al. [27] facilitates the interaction between Python and R, enabling the utilization of required Python libraries within R. Additionally, we employ the TensorFlow library Abadi et al.[10], a well-known open-source machine learning framework, to address neural network and deep learning challenges. The complete code for the solution of $y' - y = 0$ is provided in Supplementary A.1.

```
#install_tensorflow(version = "1.15.0")
library(reticulate)
library(tensorflow)
library(ggplot2)
use_condaenv('r-reticulate')
```

We initiate by outlining the structure of the neural network along with its input and output specifications. Throughout all the experiments, we employ a 6-layer neural network, each layer comprising 50 neurons.

```
tf$set_random_seed(1234)
layers <- c(1, rep(50, 5), 1) #number of layers and neurons
N_residual <- 100  #number of residuals
N_test <- 80  #length of test data
x_f <- seq(0, 1, length.out=N_residual)
x <- c(0, 1) #inputs
y <- c(1, exp(1)) # outputs
x_train <- matrix(x, nrow = 2, ncol = 1)
y_train <- matrix(y, nrow = 2, ncol = 1)
x_f_train <- matrix(x_f, nrow = length(x_f), ncol=1)
```

Subsequently, we generate weights and biases following the truncated normal distribution utilizing a Xavier initializer, referred to as the xavier_initialization function. Following this, we proceed to create a fully-connected simple deep feed-forward artificial neural network, constituting the core structure of the PINNs methodology. In this context, the hyperbolic tangent (tanh) activation function is applied in each layer, excluding the final layer.

Following the neural network's construction, we proceed to define f(t, x)

$$f \coloneqq y' - y, \tag{9}$$

and we can now approximate the *u(t, x)* with a deep feed-forward artificial neural network. Accordingly, we define the *u(t, x)* function below,

```
u(t,x) <- function(x_){
 u <- neural_network(x_, weights, biases)
return(u)
}
```

Thereby, *f(t, x)*, PINNs can be determined as,

```
f(t,x) <- function(x_){
y_ <- u_xt(x_)
y_x <- tf$gradients(y_, x_)[[1]]
# Residuals
f <- y_x - y_
return(f)
}
```

$$MSE = MSE_r + MSE_b, \tag{10}$$

and the loss function is optimized by an Adam optimizer with the following codes.

```
loss_bd <- tf$reduce_mean(tf$square(y_pred - y_train))
residual_loss <- tf$reduce_mean(tf$square(f_pred))
Adam_optim <- tf$compat$v1$train$AdamOptimizer(1e-3)
Adam_opt_train <- Adam_optim$minimize(loss_bd+residual_loss)
```

After training for 10000 epochs, the loss functions are calculated and depicted in Figure 2 as separate plots.

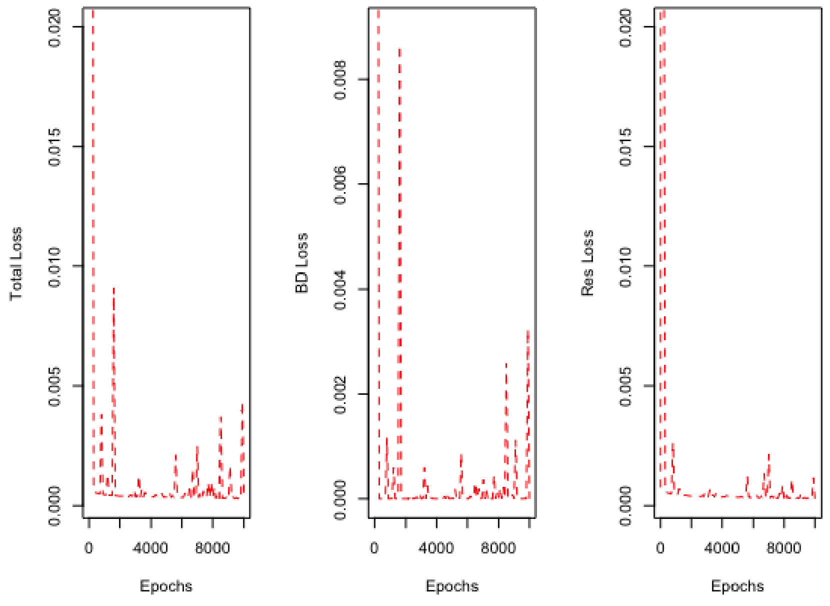


Figure 2. Total, Boundry (BD) and Residual Loss of the solution of y ′ − y = 0 with PINNs.

According to Figure 2, the Total Loss, Boundary Loss, and Residual Loss are approximately 0.00035, 0.000037, and 0.00031, respectively.

Post-training, we create a plot that showcases both the actual and estimated solutions on the test data. This comparison is made against the exact solution of the test data using the ggplot2 package Wickham and Chang]28[, as illustrated in Figure 3.

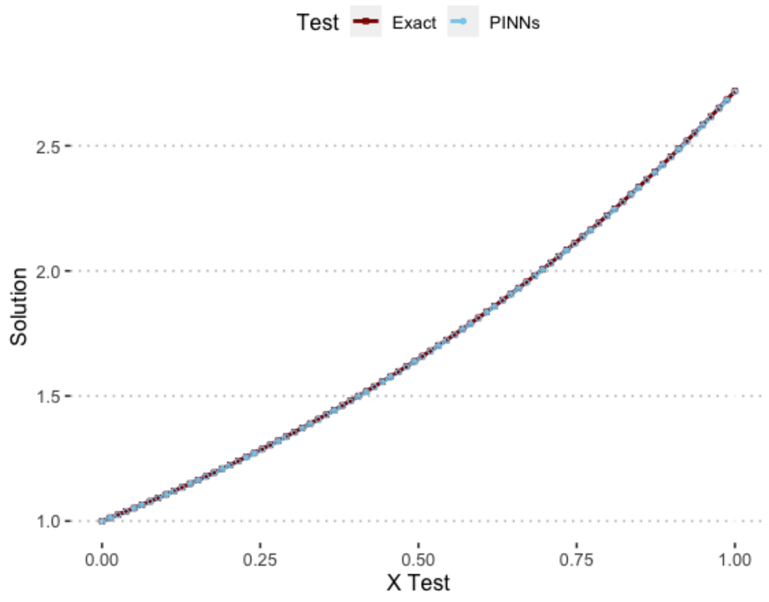


Figure 3. Comparison of predictions of test data and the exact solution.

According to Figure 3, the PINNs predicted solution accurately approximates the real solution. In conclusion, the mean squared error loss is computed from the test data and depicted in Figure 4.

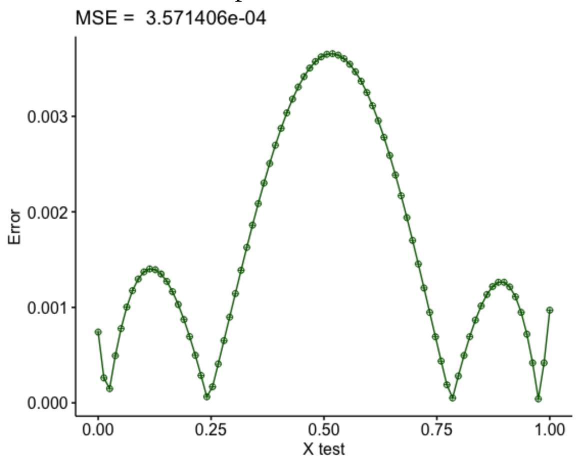


Figure 4. Mean squared error of the test data

Data-Driven Discovery of the Burgers' Equation

In this section, we tackle the one-dimensional Burgers' equation Basdevant et al. [26] using PINNs, employing the torch package Falbel and Luraschi [29] to exemplify a data-driven discovery scenario. The Burgers' equation is a fundamental partial differential equation that stems from the Navier-Stokes equations Raissi et al. [5].

Here, we demonstrate Burgers' equation first for solving forward problems using PINNs.

$$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0, x \in [-1,1], t \in [0,1] \tag{11}$$

We can define *f(t, x)* as,

$$f \coloneqq u_t + \lambda_1 u u_x - \lambda_2 u_{xx} \tag{12}$$

in which the deep feed-forward neural network approximates *u(t, x)*, and the PINNs *f(t, x)* emerges as a consequence.

We begin by installing and loading the necessary packages. The torch package is an essential open-source machine learning package developed based on PyTorch Paszke et al. [9]. All the codes for solving the Burgers' equation can be found in Supplementary A.2 on GitHub.

```
library(R.matlab)  # reading .mat data
library(pracma)
library(torch)
library(akima)
```

We start by loading the data.

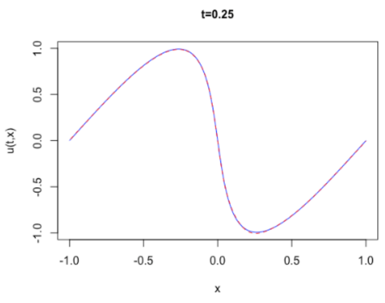
```
# Load data
data_burger <- readMat("data_burgers_shock.mat")
t <- as.vector(data_burger$t)
x <- as.vector(data_burger$x)
Exact <- t(data_burger$usol)
grid <- meshgrid(x, t)
X <- grid$X T <- grid$Y
```

The Burgers' data comprises information labeled as *t*, *x*, and *usol* in list form. We initiate by outlining the structure of the neural network along with its input and output specifications. Throughout all experiments, we employ a 9-layer neural network, each layer comprising 20 neurons.

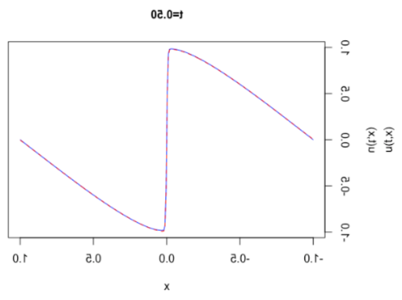
```
torch_manual_seed(1234)
nu <- 0.01 / pi N_u <- 2000
layers <- c(2, 20, 20, 20, 20, 20, 20, 20, 20, 1)
X_star <- torch_stack(c(torch_flatten(torch_tensor(X)), +
torch_flatten(torch_tensor(T))))$t()
u_star <- torch_flatten(torch_tensor(Exact))$unsqueeze(1)$t()
# Domain boundries
lb <- apply(X_star, 2, min)
ub <- apply(X_star, 2, max)
X_u_train <- X_star[id_x,]
u_train <- u_star[id_x]
u_train <- u_train + noise * torch_std(u_train) *
+ torch_randn(dim(u_train))
```

We create a training dataset containing 2000 randomly generated observations corresponding to both λ = 1.0 and λ = 0.01/π. This is done to illustrate the effectiveness of the PINNs approach. The positions of the generated training points are depicted in Figure 5(d). Following this, we update the weights and biases utilizing a simple feed-forward deep neural network structure and the LBFGS optimizer Liu and Nocedal [30], aiming to minimize the loss function. After the training process, the PINNs approach estimates both the *u(t, x)* solution of the PDEs and the parameters λ = $(\boldsymbol{\lambda_1}, \boldsymbol{\lambda_2})$ that characterize the underlying dynamics. The predictive accuracy of the PINNs approach is demonstrated in Figure 5(a-c), while the comparison between the exact and predictive outcomes for noisy and noiseless data is presented in Table 1.

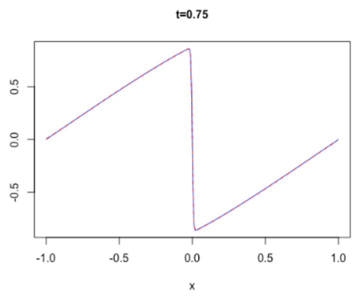
Table 1. Unequivocally indicates that the PINNs approach adeptly predicts the parameters. Notably, even with 1



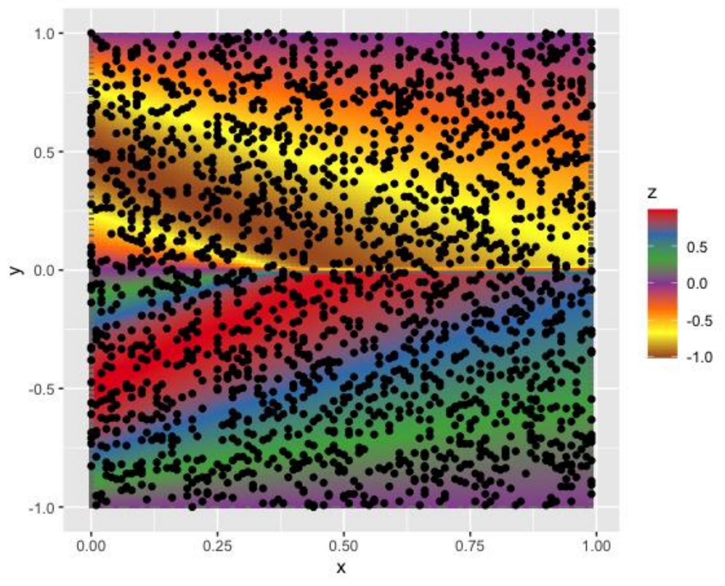
(a)



(b)



(c)



(d)

Figure 5: Burgers' equation: Exact and predicted solutions comparisons for (a) t = 0.25 (b) t = 0.50 (c) t = 0.75 and (d) predictions are given by a physics neural network of *u(t, x)* with the training data

Table 1. Correct and predicted parameters of Burgers' equation

Correct PDEs	$u_t + uu_x - 0.0031831u_{xx} = 0$
Identified PDEs (clean data)	$u_t + 0.995469uu_x - 0.0033095u_{xx} = 0$
Identified PDEs (%1 noise)	$u_t + 1.000711uu_x - 0.0031104u_{xx} = 0$

3. CONCLUSION

PINNs were first introduced by Karniadakis and his team Raissi et al. [3-5] as an innovative alternative to numerical solutions for PDEs. PINNs utilize a simple deep feed-forward neural network approach and employ automatic differentiation techniques Baydin et al. [13] to effectively address PDEs.

In this study, we have demonstrated how to apply the PINNs approach within the R programming language, leveraging the reticulate and torch packages. As far as our current knowledge extends, this study stands as the inaugural implementation of PINNs in the R programming language. Initially, we showcased the solution of a basic differential equation problem, specifically $y' - y = 0$ with conditions $y(0) = 1$ and $y(1) = e$, employing the reticulate package. We also plotted the total error, boundary error, and residual error, while additionally comparing the exact solution with predictions in a graphical representation. Our observation confirmed that the PINNs method adeptly predicts the exact solution. Finally, we calculated the MSE for the test data, resulting in an MSE of 3.571406×10^{-4}.

Subsequently, we presented a practical approach for addressing Burgers’ equation using PINNs through the PyTorch package. In this example, we contrasted the exact parameters with the predicted parameters of Burgers’ equation, considering both noisy and noiseless data. Our exploration of Burgers’ equation affirmed the PINNs approach’s capacity to accurately estimate PDE parameters. Furthermore, we designed interactive solutions for PINNs via the shiny app and the torch package, which can be accessed on the GitHub page under shiny.R.

To the best of our knowledge, this article constitutes the pioneering effort in implementing the PINNs approach within the R programming language. In our future endeavors, we aspire to adapt additional implementations of the PINNs approach to R and develop a dedicated R package for this purpose.

ACKNOWLEDGEMENTS

I would like to thank Prof. Dr. George Karniadakis for his gracious support. Dataset used in the study is available at https://github.com/melihagraz/pinns-shinny-Rcode

CONFLICT OF INTEREST

The author stated that there are no conflicts of interest regarding the publication of this article.

CRediT AUTHOR STATEMENT

Melih Ağraz: Formal analysis, Writing - original draft, Visualization, Investigation, Supervision, Conceptualization.

REFERENCES

[1] Lagaris I, Likas A, Fotiadis D. Artificial neural networks for solving ordinary and partial differential equations. IEEE Transactions on Neural Networks, 1998; 9(5): 987–1000. doi: 10.1109/72.712178.

[2] Cheng T, Lewis FL, Abu-Khalaf M. A neural network solution for fixed-final time optimal control of nonlinear systems. Automatica, 2007; 43(3): 482–490. doi: https://doi.org/10.1016/j.automatica.2006.09.021. URL https://www.sciencedirect.com/science/article/pii/S0005109806004250.

[3] Raissi M, Perdikaris P, Karniadaksi GM. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. Journal of Computational Physics, 2019; pages 686–707.

[4] Raissi M, Perdikaris P, Karniadaksi GM. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations. arXiv preprint. arXiv:1711.10561. 2017a.

[5] Raissi M, Perdikaris P, Karniadaksi GM. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations. arXiv preprint. preprint arXiv:1711.10566. 2017b.

[6] Soetaert K, Petzoldt T, Setzer RW. Solving differential equations in R: Package deSolve. Journal of Statistical Software, 2010; 33(9): 1–25. doi: 10.18637/jss.v033.i09.

[7] Soetaert K, Meysman F. R-package reactran : Reactive transport modelling in r. 2010.

[8] Soetaert K. rootSolve: Nonlinear root finding, equilibrium and steady-state analysis of ordinary differential equations. R package 1.6. 2009.

[9] Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, et al. Pytorch: An imperative style, high-performance deep learning library. In: Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, Garnett R, editors. Advances in Neural Information Processing Systems 32. Curran Associates, Inc.; 2019. pages 8024–8035.

[10] Abadi M, Agarwal A, Barham P, Brevdo E, Chen Z, Citro C, et al. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

[11] Markidis S. The old and the new: Can physics-informed deep-learning replace traditional linear solvers? Frontiers in big Data, 2021; 4: 669097.

[12] Cai S, Wang Z, Wang S, Perdikaris P, Karniadakis G. Physics-informed neural networks (pinns) for heat transfer problems. Journal of Heat Transfer, 2021; 143. doi: 10.1115/1.4050542.

[13] Baydin A, Pearlmutter B, Radul A, Siskind J. Automatic differentiation in machine learning: A survey. Journal of Machine Learning Research, 2018; 18: 1–43.

[14] Kingma D, Ba J. Adam: A method for stochastic optimization. International Conference on Learning Representations, 2014.

[15] Zhang D, Lu L, Guo L, Karniadakis G. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. Journal of Computational Physics, 2019; 397. doi: 10.1016/j.jcp.2019.07.048.

[16] Pang G, Lu L, Karniadakis G. fpinns: Fractional physics-informed neural networks. SIAM Journal on Scientific Computing, 2019; 41: A2603–A2626. doi: 10.1137/18M1229845.

[17] Jagtap A, Kharazmi E, Karniadakis G. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. Computer Methods in Applied Mechanics and Engineering, 2020; 365: 113028. doi: 10.1016/j.cma.2020.113028.

[18] Meng X, Li Z, Zhang D, Karniadakis G. Ppinn: Parareal physics-informed neural network for time-dependent pdes. Computer Methods in Applied Mechanics and Engineering, 2020; 370: 113250. doi: 10.1016/j.cma.2020.113250.

[19] Jagtap A, Karniadakis G. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. Communications in Computational Physics, 2020; 28: 2002–2041. doi: 10.4208/cicp.OA2020-0164.

[20] Pang G, D'Elia M, Parks M, Karniadakis G. npinns: Nonlocal physics-informed neural networks for a parametrized nonlocal universal laplacian operator. algorithms and applications. Journal of Computational Physics, 2020; 422: 109760. doi: 10.1016/j.jcp.2020.109760.

[21] Kharazmi E, Zhang Z, Karniadakis GE. hp-vpinns: Variational physics-informed neural networks with domain decomposition. Computer Methods in Applied Mechanics and Engineering, 2021; 374.

[22] Shukla K, Jagtap AD, Karniadakis GE. Parallel physics-informed neural networks via domain decomposition. Journal of Computational Physics, 2021; 447: 110683.

[23] Yang L, Meng X, Karniadakis GE. B-pinns: Bayesian physics-informed neural networks for forward and inverse pde problems with noisy data. Journal of Computational Physics, 2021; 425: 109913.

[24] Lu L, Jin P, Pang G, Zang H, Karniadakis G. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. Nature Machine Intelligence, 2021; 3: 218–229. doi: 10.1038/s42256-021-00302-5.

[25] Raissi M, Karniadakis GE. Hidden physics models: Machine learning of nonlinear partial differential equations. Journal of Computational Physics, 2018; 357: 125–141.

[26] Basdevant C, Deville M, Haldenwang P, Lacroix J, Ouazzani J, Peyret R, et al. Spectral and finite difference solutions of the burgers equation. Computational Fluid Dynamics. 1986; pages 23–41.

[27] Ushey K, Allaire J, Tang Y. reticulate: Interface to 'Python', 2022. URL https://rstudio.github.io/reticulate/, https://github.com/rstudio/reticulate.

[28] Wickham H, Chang W. Ggplot2: Create Elegant Data Visualisations Using the Grammar of Graphics, 2016. URL URLhttps://CRAN.R-project.org/package=ggplot2.

[29] Falbel D, Luraschi J. torch: Tensors and Neural Networks with 'GPU' Acceleration, 2022. URL https://torch.mlverse.org/docs,https://github.com/mlverse/torch.

[30] Liu DC, Nocedal J. On the limited memory bfgs method for large scale optimization. Mathematical Programming, 1989; 45: 503–528.

[31] Ruder S. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747. 2016.

[32] Agraz, M. Evaluating single multiplicative neuron models in physics-informed neural networks for differential equations. Scientific Reports, 2024, 14(1), 19073.