



AMITY UNIVERSITY

Project Report

NTCC In-House Project (ETPT100)

Solving Partial Differential Equations using Deep Learning Techniques

Submitted By

Nayan Ranjan Das
Enrolment No.: A2305220148
5CSE2, B.Tech. (2020-24)

Faculty Guide

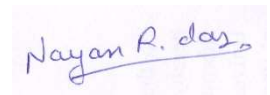
Dr. Garima Mahendru
Faculty, Department of Computer
Science and Engineering

DECLARATION BY THE STUDENT

I, **Nayan Ranjan Das**, student of B.Tech. (CSE) hereby declare that the project titled “**Solving Partial Differential Equations using Deep Learning Techniques**” which is submitted by me to Department of Computer Science and Engineering, ASET, Noida, Amity University Uttar Pradesh, in partial fulfilment of requirement for the award of the degree of Bachelor of Technology (CSE), has not been previously formed the basis for the award of any degree, diploma or other similar title or recognition.

The Author attests that permission has been obtained for the use of any copy righted material appearing in the report other than brief excerpts requiring only proper acknowledgement in scholarly writing and all such use is acknowledged.

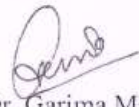
Date: July, 24 2022

A handwritten signature in blue ink that reads "Nayan R. Das" with a stylized flourish at the end.

Signature

Certificate

This is to certify that Mr. Nayan Ranjan Das, student of B.Tech. (CSE) has carried out the work presented in the project of the Term Paper entitled “Solving Differential Equations using Deep Learning Techniques” as a part of Third year programme of Bachelor of Technology (CSE) from Amity School of Engineering and Technology, Noida, Amity University, Uttar Pradesh under my supervision.



Dr. Garima Mahendru

Amity School of Engineering and Technology, AUUP

22/7/22

Table of Contents

1. Introduction	1
2. Literature Review	4
3. Methodology.....	6
4. Result.....	9
5. Conclusion.....	10

Abstract

In this project I have proposed a simple Artificial Neural Network (ANN) framework which is used to solve IVP and BVP problems for PDEs. Since, there are many hefty models available for solving such problems, this project is created with the intention of creating a computationally minimalistic framework which can solve any PDEs provided their constraints and the form of solution is known. A labelled master dataset is constructed on the domain using evenly spaced vectors, from which, samples were randomly selected to create the training set, while fixing the size of each training set. The ANN sequential model created using **Tensorflow [1] and Keras [2]** employs the cost function, the optimization procedure and the architecture posterior to meticulous experimenting with various hyperparameters. The model is trained for a set number of epochs and for a particular sample size with a view to analyse and demonstrate their impact on the result and to further enhance the model. The result is displayed via comparative plots to view the learning capabilities of the model. The time taken by a framework for learning and the mean error in each model is also tabulated for analysis of each combination of number of epochs and size of training set. The review of publications for the project is presented with discussions on improvements and other techniques by researchers in these fields and further improvements and implementation of new methods on this project are promised.

1. Introduction

Equations which establish a relationship between an function and its derivatives with respect to its independent variables is called a Differential equation. When the unknown function is a multivariate function and the equation contains its partial derivatives, then the differential equation is called Partial Differential Equation (PDE). Their solutions are determined using various methods. PDEs are clustered into certain types on the basis of their structure. This includes linear, non-linear, homogenous and non-homogenous. Among these, non-linear PDE exhibit very complex behaviour and model several natural phenomena which may be seen to layman eyes as chaos. Different types of PDEs have different methods for determining their solution. For instance, Lagrange method is used for solving linear PDEs and Charpit method is applied on non-linear PDEs. Another method called "separation of variables" is used to solve for functions that are sum or products of single-variable functions. This is applied on linear homogeneous PDEs. The general form of solution is $u = X(x)T(t)$ and is used for wave equation, heat equation, Laplace equation, Helmholtz equations and several other common forms of equations.

There is a wide range of applications for PDEs in physics and engineering. These generally include modelling scientific processes to analyse their effect in certain conditions. Since, a variety of physical processes are controlled by known PDEs, solving for their solution becomes important for studying them. For example, the diffusion process is described by the Heat equation defined by Joseph Fourier, the application of Navier-Stokes and Burgers' equations in fluid dynamics and other considerable applications in the field of engineering like traffic flow, elasticity, harmonics and so on. Among other fields where PDEs are implemented biology has many modelling problems like monitoring biological and chemical oscillations with the Kuramoto model, the McKendrick-von Foerster equation for age modelling and cell proliferation. Even the commonly used logistic or sigmoid function has its roots with Verhulst equation describing the biological population growth and was used for modelling COVID-19 cases in the early phases of the pandemic. Interestingly, the world of economics and finance is also familiar with the impact of PDEs as equations like Black-Scholes and Fokker-Planck are used extensively to model the market growth. Since, PDEs are such an indispensable component for so many fields and disciplines, studying them and their solutions is imperative.

The development of ANN is among the most crucial advancements in the field of Machine Learning and AI. The schema is inspired by the human brain. where billions of neurons fire constantly to help in making the decisions while we get around to accomplish our everyday tasks. The architecture of ANN is fashioned in a similar way, consisting of layers of neurons. The first layer receives the data from the developer. This can either be

well labelled data for supervised techniques or unlabelled and unfiltered data for the application of unsupervised methods, it entirely depends on the requirement. The hidden layers are layers of neurons or nodes stacked between input layer and the output layer. These help the model to learn various aspects of the data and ultimately a model is developed which gives outputs with least possible error and highest possible accuracy for given constraints. The output layer is where the nodes receive the final values which in fact are the predicted results. But the architecture alone cannot enable a model to achieve machine learning capabilities. Thus, it requires other features. To help ANN learn linear and non-linear trends in data, linear and non-linear properties must be inculcated in the model. The sum is the sum of products of incoming values and the weight parameters is called the weighted sum. This helps teaching linear relationships to the model. This weighted sum is then put in a non-linear function which provides the model with non-linear properties. These are called activation functions. There are traditional tanh and sigmoid functions, as well as new functions like Rectified Linear Unit (ReLU), Leaky ReLU, Softmax, etc. are also implemented heavily as per requirement.

Now, although the model might be ready to give outputs on the supplied input data, there would not be any learning taking place. We implement the concept of optimization for making the ANN framework to start learning and start giving better results with each iteration. For that, we define a loss function which best suits our need. The loss function would define how far away the predicted solution is from the actual solution. There are several classes of loss functions available which can be tailored and altered as required. The frequently used cost functions for regression problems are Mean Squared Loss, Mean Absolute Loss, Sum of Squared Loss, etc., while Binary Cross-Entropy and Sparse Cross-Entropy are used for classification problems. The ultimate loss value for an iteration is then optimized using an optimizing algorithm. There is a huge library of algorithms. Most of these are Gradient Descent based algorithms. Thus, a need arises to calculate the gradient of the cost function in regards to the parameters. Here, the concept of Backpropagation comes to rescue. It is method to calculate the required gradients. For calculation, it enlists the help of **Automatic Differentiation [3]** algorithm. Auto-differentiation is such an integral part of Neural Networks modelling that all popular ANN environment support this feature. For instance, in **Tensorflow [1]**, the implementation is provided with GradientTape() (explored briefly in Methodology). Thus, with Backpropagation and continuous optimization of loss function for the weight matrices, the ANN model should give excellent predicted results provided appropriate number of iterations take place.

Although, a simple Multi-Perceptron ANN model would give very good results, complex problems require a lot of computation and thus they require several hidden layers as one or two layers may not suffice the computational requirement for the problem. Thus, implementation of multiple layers containing nodes capable of processing data in non-

linear fashion is called Deep Learning. It is a promising implementation of AI and is capable of achieving levels of accuracy other models may fail to reach. With technological advancements in the field of processors with the use of high-speed GPUs and easy-to-use ANN environments, Deep Learning is able to achieve wonders. Some recent developments in the field are Convolutional Neural Network (CNN), Recurrent Neural Network (RNN) and Modular Neural Network.

In this project I have created a simple and minimalistic ANN model to solve PDE and demonstrate the model on classical wave equation. The model is trained using evenly spaced vector created on the domain for independent variables and the training dataset is generated via the actual solution of the equation. We test the framework for an array of samples and for different values of epochs for training and describe the acquired outcome with the aid of plots and tables.

2. Literature Review

Lagaris et. al. present it as an optimization problem in [1]. The model put forth conjectured that the arithmetic difference between the differential obtained from actual solution and from the function definition be minimized over discrete values of domain set to generate predicted solutions with great accuracy. The components for initial values and boundary values were added separately to the trial solution formula to train the model. They preferred a Quasi-Newton BFGS algorithm to optimize the loss function, which was chosen by **Raissi et. al.** as well in [5]. They presented the exactitude of the anticipated solution with comparison to Finite Element Method (FEM), a numerical method for solving PDEs. The best aspect of this approach is that there is no need for any labelled dataset containing solutions of differential equations at various points as this model gets its set of actual solutions from the differential function.

Another paper [5] that took the field to new levels was **Raissi et. al.** where they proposed a Physics Informed Neural Network, a supervised learning approach where an ANN model which is capable of solving any PDE while respecting the laws established by it. It exploits the **Universal Approximation Theorem** [6] and the concept of **automatic differentiation** [3]. They obtain a dataset using spectral methods to solve the PDEs. The main objective is to optimize the cost function which is a blend of several terms of Mean Squared Error concerning various points over the domain. These terms include error of solutions predicted by the framework for the function providing with the physical information with the value of differential equation at various points of training set. They have tabulated error with respect to number of points used for training and number of collocation points used for minimizing the differential equation function. The examples of PDEs used to demonstrate the approach are Burgers' Equation and Schrodinger's equation. The same approach was proposed by **Raissi** in another paper [7] where the model was put into effect for some other examples of Burgers' Equation and Korteweg-de Vries equation, Kuramoto-Sivashinsky equation, Nonlinear Schrodinger equation. The author also proposed its applications in stochastic PDEs and Geostationary Operational Environmental Satellites (GOES) data concerning seas surface temperatures.

There have been some of the published applications of PINN approach proposed by Raissi et.al. The paper [8] where the authors (**Moseley et. al.**) implement the PINN model on 2D Acoustic Wave Equation. They acquire the data from Finite Difference Method (FDM). The model is tested for different values of governing parameters like density and velocity. Although, this framework allowed them to infer physical phenomena like transmission, reflection, compression and expansion of waves in various interfaces, they observed a limitation: the PINN models need to be trained again for any new set of constraints. Another instance of implementation of PINN for wave equations

is **Guo et. al.** [9] where they tested this for 1D wave equation, Korteweg-de Vries equation and KdV-Burgers' equation and achieved good results.

Among other applications, **Rao et. al.** in [10] have presented a PINN model for solving elastodynamics problems without using any labelled data. They propose a mixed variable scheme for their framework and this supported their hypothesis and drastically improved the accuracy and learning capabilities of the model. They performed test on a plate under tension, and several 2D wave propagation on various boundaries.

There has been success in other approaches which do not concern PINN approach. This includes a comprehensive study [11] by **Beck et. al.** where several different Deep Learning based approximation techniques for linear and non-linear PDEs are presented. Another paper [12] by **Sirignano et. al.** which presents a Deep-Learning algorithm - DGM for solving PDE greatly resembles [4] by **Lagaris et. al.** and [5] by **Raissi et. al.** approaches. Along with that the paper also presents a series of numerical methods for solving Differential Equations, like a more computationally cheaper Monte-Carlo approximation for second derivative. They test their algorithm on High-dimensional Hamilton-Jacobi-Bellman PDE, Burgers' equation and discusses other Neural Network approximation theorems.

There are several environments available for scientific computing and for solving Differential Equations. Also, there are packages being established for popular Neural Network environments. One among these is SciANN [13], introduced by **Haghighat et. al.** at MIT, Cambridge. This package provides for scientific computing and more specifically, Physics Informed Deep Learning. It works as a wrapper over Google's **Tensorflow** [1] and François Chollet's **Keras** [2].

3. Methodology

I have created a model for predicting the solution of a PDE with given initial values and boundary values. I started with obtaining a labelled dataset for the training of the ANN model. The model is then implemented on the dataset (or a subset of it) to get series of predicted values of the solution for the PDE for the dataset.

The PDE problem in question is:

$$\frac{\partial^2 u(x, t)}{\partial t^2} = c^2 \frac{\partial^2 u(x, t)}{\partial x^2}, \quad x \in [0, 1], \quad t \in [0, 1]$$

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

$$u(x, 0) = u_0 \sin^2\left(\frac{\pi x}{l}\right)$$

$$\frac{\partial u}{\partial t}_{t=0} = 0$$

The constants have the following values

$$c = 1, \quad l = 1, \quad u_0 = 1$$

The solution acquired analytically is:

$$u(x, t) = \frac{3u_0}{4} \sin\left(\frac{\pi x}{l}\right) \cos\left(\frac{\pi ct}{l}\right) - \frac{u_0}{4} \sin\left(\frac{3\pi x}{l}\right) \cos\left(\frac{3\pi ct}{l}\right)$$

The dataset is prepared using **Numpy** [14], a Python Library for mathematical functions. The *numpy.linspace()* function is used to create a vector of 200 evenly spaced points between the boundaries of the concerned equation (here, $x, t \in [0, 1]$). The independent variables x and t attain their values from this vector.

```
1 x = np.linspace(0,1,200)
2 t = np.linspace(0,1,200)
```

These independent variables vectors are used to get the actual solution for the points (x, t) using the analytically solved solution for the equation. The obtained solution is a 200x200 matrix containing values of the solution for each combination of values of x and t .

```

1 def get_wave(x,t):
2     x_t_list = []
3     list_of_lines = []
4     for t_val in t:
5         one_line = []
6         # for storing values of u(x,t) for a particular t_val
7         for x_val in x:
8             A = (3*y0/4) * np.sin(pi*x_val/l) * np.cos(pi*c*t_val/l)
9             B = (y0/4) * np.sin(3*pi*x_val/l) * np.cos(3*pi*c*t_val/l)
10            one_line.append(A-B)
11            x_t_list.append([x_val, t_val])
12            # since all u(x,t) for particular t_val are computed
13            # the list is appended
14            list_of_lines.append(one_line)
15        u = np.array(list_of_lines)
16        x_t = np.array(x_t_list)
17        return u, x_t

```

For the training dataset I have randomly chosen 'n' number of samples from the main dataset. The size of training set is kept a variable so that the effect of number of samples on the accuracy of the prediction can be studied for some given number of epochs the model will iterate over the values for optimization. The function to define training set for 'n' samples is:

```

1 def get_training_data(samples):
2     index = randint(0,40000-1)
3     u_data = u_new[index]
4     x_t_data = x_t[index]
5
6     for i in range(1,samples):
7         index = randint(0,40000-1)
8         u_data = np.vstack((u_data, u_new[index]))
9         x_t_data = np.vstack((x_t_data, x_t[index]))
10    x_t_data.shape

```

I have used **Tensorflow** [1] and **Keras** [2] Python packages for developing the model. The Multilayer Perceptron or the ANN Sequential framework consists of 2 hidden Dense layers, each having 20 neurons.

```

1 nodes = 20
2 NN = keras.Sequential([
3     keras.layers.Dense(nodes,activation=keras.activations.tanh,
4                          input_shape=(2,)),
5     keras.layers.Dense(nodes,activation=keras.activations.tanh),
6     keras.layers.Dense(1,activation=keras.activations.tanh)
7 ])
8
9 NN.compile(
10    optimizer=keras.optimizers.RMSprop(learning_rate=0.01),
11    loss=keras.losses.MeanSquaredError(),
12 )

```

These layers help in computing the predicted value of solution by using Hyperbolic Tangent function as the activation function given by:

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}}$$

The neurons multiply and add the incoming values with the associated weights providing the linear aspect of Neural Networks which is then used as the input for the activation function to provide the non-linearity required for generating the solution. The predicted solution is collected in the output node of the model. This output is then compared with the actual solution to give an error value which is to be minimized.

Other hyperparameters set for the model are the loss function, the optimizer and the learning rate for the optimizing algorithm. I have selected the Mean Squared Error as the loss function.

$$MSE = \frac{1}{N} \sum_{i=1}^N |u_i - \hat{u}_i|^2$$

With meticulous cut and try method I have selected **RMSprop** [15] optimizing algorithm as it was able to minimize the loss function with least number of epochs. Stochastic Gradient Descent and Adam and were in the group of algorithms that were tested for the model. The learning rate for RMSprop was selected in a similar manner and was finalized at 0.01.

For testing the accuracy, the plots for actual solution and predicted solution are presented over the same axis for various instances of time, so that the results can be compared visually.

```
1 start = time.time()
2 history = NN.fit(x_t_data, u_data, verbose=0, epochs=epochs)
3 elapsed = time.time() - start
4
5 print(f"first loss = {history.history['loss'][0]}")
6 print(f"last loss = {history.history['loss'][-1]}")
7 print(f"time elapsed = {elapsed:.3f} sec")
```

4. Result

The effectuation of the model is primarily assessed by plotting the curve of true solution and the predicted solution for a series of time variable, training set sizes and epochs. The plots are for solution at time $t=0.10$ and 0.90 , for 100, 500, 1000 and 5000 samples of training set and for 500 and 1000 epochs. The plots are created using **Matplotlib** [16] Python Library. The inference gathered from the variety of plots is that there is virtually no effect of increasing the number of epochs and the training set size on the result. The plots for 500 epochs and 1000 epochs, the plots for 100 samples and 5000 samples, all are virtually the same. The difference is very minute and can be noticed when the loss values of the models for various training sets and number of epochs are tabulated and compared. The least loss is observed for training set with 5000 samples.

Table 4.1 Relative L_2 Error Values for different training set sizes and epochs

Training Set Size	500 epochs	1000 epochs
100	1.79e-02	7.51e-03
500	2e-03	2.66e-03
1000	2.17e-03	1.82e-03
5000	1.36e-03	1.57e-03

Table 4.2 Time taken (in seconds) by models for training

Training Set Size	500 epochs	1000 epochs
100	3.136	6.500
500	9.821	18.659
1000	17.696	30.323
5000	73.918	135.287

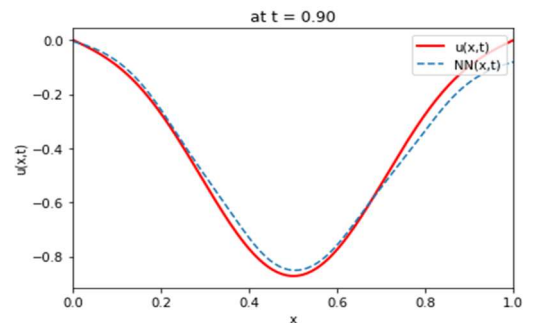
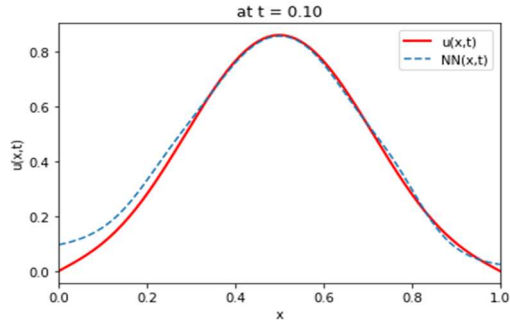


Figure 4.1 Comparison of solutions predicted by the proposed model (blue dashed) and the actual solution (red) over the domain at time $t=0.10$ and 0.90 for 500 epochs and 100 training samples

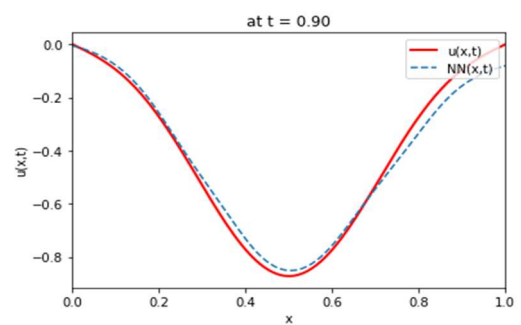
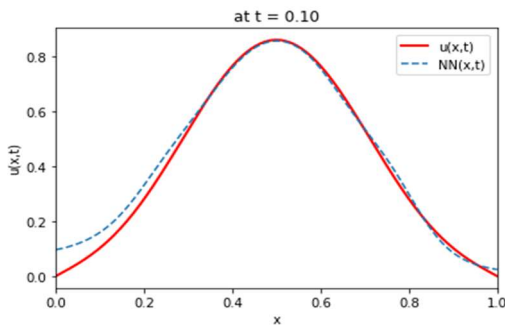


Figure 4.2 Comparison of solutions predicted by the proposed model (blue dashed) and the actual solution (red) over the domain at time $t=0.10$ and 0.90 for 1000 epochs and 5000 training samples

5. Conclusion

The proposed model can predict solution for the given wave equation with fair accuracy, considering the limited number of iterations. The constructed Neural Network model is able to predict the solution with a least relative L_2 error of $1.36e-03$ at 1000 epochs over 5000 samples. Using this, we can say that although with usage of complex approaches proposed by Raissi et. al. [5] and Moseley et. al. [8] can improve the physical learning of the model, relatively simple and computationally cheaper techniques are able to predict the solution with appreciable accuracy as well.

There are several areas where there is scope of improvement. This includes application of PINN method for a variety of Wave Equation problems. PINN is among the series of latest advancements in the field of Neural Networks used for solving complex PDEs. Further improvement can be achieved with application of Monte-Carlo approximation coupled with Neural Networks approach. For representation of result, the predicted solution can be compared with solution obtained from numerical methods like Finite Difference Method, Finite Element Method or other known spectral methods. After said improvements, the model can then be applied for solving 2D, 3D Wave Equations with minimal set of training points, minimal architecture and other complex equations like Fokker-Planck equation, Schrodinger's equation, Navier-Stokes equation, etc.

References

- [1] Abadi, Martín, et al. "Tensorflow: Large-scale machine learning on heterogeneous distributed systems." *arXiv preprint arXiv:1603.04467* (2016).
- [2] Keras-Team. "Keras-Team/Keras: Deep Learning for Humans." GitHub, www.github.com/keras-team/keras.
- [3] Baydin, Atilim Gunes, et al. "Automatic differentiation in machine learning: a survey." *Journal of Machine Learning Research* 18 (2018): 1-43.
- [4] Lagaris, Isaac E., Aristidis Likas, and Dimitrios I. Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations." *IEEE transactions on neural networks* 9.5 (1998): 987-1000.
- [5] Raissi, Maziar, Paris Perdikaris, and George Em Karniadakis. "Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations." *arXiv preprint arXiv:1711.10561* (2017).
- [6] "Universal Approximation Theorem - Wikipedia". *Wikipedia*, en.wikipedia.org/wiki/Universal_approximation_theorem. Accessed 17 July 2022.
- [7] Raissi, Maziar. "Deep hidden physics models: Deep learning of nonlinear partial differential equations." *The Journal of Machine Learning Research* 19.1 (2018): 932-955.
- [8] Moseley, Ben, Andrew Markham, and Tarje Nissen-Meyer. "Solving the wave equation with physics-informed deep learning." *arXiv preprint arXiv:2006.11894* (2020).
- [9] Guo, Yanan, et al. "Solving partial differential equations using deep learning and physical constraints." *Applied Sciences* 10.17 (2020): 5917.
- [10] Rao, Chengping, Hao Sun, and Yang Liu. "Physics informed deep learning for computational elastodynamics without labeled data." *arXiv preprint arXiv:2006.08472* (2020).
- [11] Beck, Christian, et al. "An overview on deep learning-based approximation methods for partial differential equations." *arXiv preprint arXiv:2012.12348* (2020).
- [12] Sirignano, Justin, and Konstantinos Spiliopoulos. "DGM: A deep learning algorithm for solving partial differential equations." *Journal of computational physics* 375 (2018): 1339-1364.
- [13] Haghighat, Ehsan, and Ruben Juanes. "Sciann: A keras/tensorflow wrapper for scientific computations and physics-informed deep learning using artificial neural networks." *Computer Methods in Applied Mechanics and Engineering* 373 (2021): 113552.
- [14] Harris, Charles R., et al. "Array programming with NumPy." *Nature* 585.7825 (2020): 357-362.
- [15] Hinton, G. "Neural Networks for Machine Learning Lecture 6a Overview of Mini-Batch Gradient-Descent". *Coursera*, www.cs.toronto.edu/~tijmen/csc321/slides/lecture_slides_lec6.pdf. Accessed 17 July 2022.
- [16] Hunter, John D. "Matplotlib: A 2D graphics environment." *Computing in science & engineering* 9.03 (2007): 90-95.