Deep Learning for Approximating Solutions to Partial Differential Equations

ID: 209537475
Department of Computer Science
University of Bath
`Contribution: 100%`

ID: 209609303
Department of Computer Science
University of Bath
`Contribution: 100%`

ID: 2094548841
Department of Computer Science
University of Bath
`Contribution: 100%`

ID: 209548150
Department of Computer Science
University of Bath
`Contribution: 100%`

May 7th, 2021

Abstract

Modelling of physical systems is crucial to many industries. Many physical systems can be described by systems of Partial Differential Equations (PDE). This paper aims to implement a Physics Informed Neural Network (PINN) based on the methodology of Raissi et al [1]. The PINN was applied to approximate the solution to the Burgers Equation as a validation of the implementation. This PINN was then applied to the heat equation as it is one of the most commonly modelled PDEs in industry [2]. Lastly, PINN was extended to a higher dimension by applying it to the time-dependent Heat equation in two spatial-dimensions. These approximate solutions were then compared and evaluated against their respective Finite Element Analysis approximation using the FEniCS Python library.

Contents

1 Introduction

Physics and the mathematical laws that define it govern from the gravitational laws and the movement of galaxies, to the behaviour of particles in solids, liquids and gases, and down to the subatomic scale defined by the laws of Quantum Mechanics. What all these systems have in common is that they develop over space-time. Partial Differential Equations (PDEs) allow for the description of these dynamic physical systems, allowing us to delve deeper into understanding the fundamental laws that surround us. Solving these systems using neural networks is challenging due to the complexity of the loss functions required and formulation of the input data. Furthermore by answering this question it allows us to learn how these networks perform when compared to more conventional methods such as Finite Element Methods. Unlike traditional uses of neural networks, such as classification, we wish to contribute to the field of deep learning by pushing the boundaries of the problems in which neural networks can solve.

1.1 Motivation and Aims

The understanding of physical systems allows us to better predict and thus harness the environment around us, allowing for the development of many modern conveniences. PDEs often describe these dynamic physical systems, including fluid flow, heat transfer, and chemical dispersion. Conventional methods for solving PDEs, like the Finite Element Method (FEM) or Runge-Kutta Fehlberg (RKF), can be computationally expensive and time consuming for complex equations. Many numerical methods begin to break down when applied to high dimensional problems due to the curse of dimensionality [3]. In addition, these methods are limited to discrete solutions and often suffer from higher deviation error in the solutions [4].

This project involved recreating the results from Raissi et al. [1] as validation that the deep neural network developed worked as expected. The neural network was then applied to solve for the heat equation in one spatial dimension with time and two spatial dimensions with no time dependency. Lastly, we approximated this 2-D spatial solution with a time dependency problem, again using the corresponding heat equation, and further exploring how the network would perform in this higher dimensionality regime.

2 Background

2.1 Physics Informed Neural Networks

Following Raissi et al. [1], data-driven solutions were computed to PDEs with a general form (1), where $u(t, x)$ stands for the hidden solution and $\mathcal{N}[\cdot]$ is a nonlinear differential operator. The PDEs under investigation fall into the category of continuous time models, where we define $f(t, x)$ to be left hand side of the equation (1), and approximate $u(t, x)$ by a deep neural network, thereby resulting in a Physics Informed Neural Network $f(t, x)$ (PINN) [1].

$$u_t + \mathcal{N}[u] = 0 \tag{1}$$

In this investigation two particular PDEs were studied: The heat equation and The Burgers Equation. Implemented in previous literature [1] [5], Burgers equation served as a basis to understand how the structure of a PINN works. The gained knowledge of the network's architecture was then applied to the corresponding heat equation, a relatively easier PDE to solve for and analyse, allowing to challenge the complexity involving a higher dimensionality.

2.2 The Heat Equation

This equation governs the temperature distribution in any object. Its derivation arises from a simple uniform rod with a non-uniform temperature distribution, with thermal energy (heat) being transferred from hot to cold temperature regions [6]. By conservation of energy one can obtain the 1-D solution to the heat equation which can be extended to the vector solution (2) by using the gradient operator $\left(\triangledown = \frac{\partial u}{\partial x} + \frac{\partial u}{\partial y} + \frac{\partial u}{\partial z}\right)$, where κ is the thermal diffusivity of a material with units $\frac{L^2}{T}$.

$$\frac{\partial u}{\partial t} - \kappa \triangledown^2 u = 0 \tag{2}$$

2.3 The Burgers Equation

Studied by Johannes Martinus Burgers in 1948 as a model for turbulence, this is a fundamental equation (3) that derives from the Navier-Stokes equations of fluid motion for nonlinear diffusion, yet it is only one space dimensional and lacks the pressure gradient driving the flow. This PDE can be found in many areas of mathematics, from fluid mechanics and nonlinear acoustics, to gas dynamics and traffic flow [7]. The equation combines two areas of study: on the left, the momentum flow associated with conservation laws and, on the right, the diffusion associated with the heat equation, from the kinematic viscosity (υ) of the fluid in question, with units $\frac{L^2}{T}$.

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \upsilon\frac{\partial^2 u}{\partial x^2} \tag{3}$$

2.4 Finite Element Analysis

One of the more common methods for approximating solutions to PDEs, and for simulating physical systems in solid mechanics is Finite Element Analysis (FEA) [2]. FEA applies finite element discretization by introducing a triangular mesh grid, which the PDE is evaluated on [8]. The FEniCS *Python* library [9] was used for comparison of an alternative method to solving PDEs and for evaluation of the results. This library allows for simple evaluation of the PDE by allowing the user to input the PDE and its boundary conditions in its variational form.

3 Methodology

3.1 Loss Formulation

A data-driven approach to approximating the solution to a given PDE can be formulated by encoding the PDE into the loss function of a neural network. This allows the neural network to constrain the possible search space to a more manageable size, which allows for the successful mapping of the coordinates to its respective solution [1]. To explain the loss formulation, we will consider an example, a PINN solving the heat equation in one spatial dimension over time, starting at a constant temperature. Recall equation 2 with the following boundary (5) and initial conditions (6), where T_0 is the initial temperature.

$$\frac{\partial T}{\partial t} - k\nabla^2 T = f(x,t) \tag{4}$$

$$T(0,t) = T(1,t) = 0 \tag{5}$$

$$T(x,0) = T_0 \tag{6}$$

The solution to a PDE must satisfy the equations for which it is governed by. Using a neural network allows for the solution to be formulated as an optimisation problem. The network optimizes its prediction by how well it satisfies the differential equation and its boundary/initial conditions. This can be seen below, where each equation is given a new variable. Note that in equation (7) a rearrangement is performed, setting the equation equal to zero. This is the basis for constraining the solution within the domain, and is what allows the neural network to learn the solution to the PDE within the boundaries without the exact solution. However, at the boundary and initial conditions, the network learns the parameters that output values to satisfy these conditions (8) & (9).

$$D = \frac{\partial T}{\partial t} - k\nabla^2 T - f(x,t) = 0 \tag{7}$$

$$BC = T(0,t) = T(1,t) = 0,\ \forall\ t \in [0,5] \tag{8}$$

$$IC = T(x,0) = T_0, \forall\ x \tag{9}$$

The loss function of the neural network is then formulated as:

$$Loss_{NN} = MSE_D + MSE_{BC} + MSE_{IC} \tag{10}$$

where,

$$MSE_D = \frac{1}{N_D}\sum_{n=1}^{N_D}(\hat{D_n} - D_n)^2 = \frac{1}{N_D}\sum_{n=1}^{N_D}(\hat{D_n} - 0)^2 \tag{11}$$

$$MSE_{BC} = \frac{1}{N_{BC}}\sum_{n=1}^{N_{BC}}(B\hat{C}_n - BC_n)^2 = \frac{1}{N_{BC}}\sum_{n=1}^{N_{BC}}(B\hat{C}_n - 0)^2 \tag{12}$$

$$MSE_{IC} = \frac{1}{N_{IC}}\sum_{n=1}^{N_{IC}}(I\hat{C}_n - IC_n)^2 = \frac{1}{N_{IC}}\sum_{n=1}^{N_{IC}}(I\hat{C}_n - 0)^2 \tag{13}$$

The use of the loss MSE_D constrains the solution in the domain, while MSE_{BC} constrains the solution at the boundary conditions, and MSE_{IC} at the initial conditions. Similar to the majority of modern deep learning applications, back-propagation is used to adjust the neural network parameters using $Loss_{NN}$ as the cost function. Using this combined loss allows the neural network to best learn the mapping from input variables to the solution of the given PDE that satisfies boundary and/or initial equations. In the heat equation this translates to approximating a value T given x and t as the input variables, that satisfies equations 7, 8 and 9.

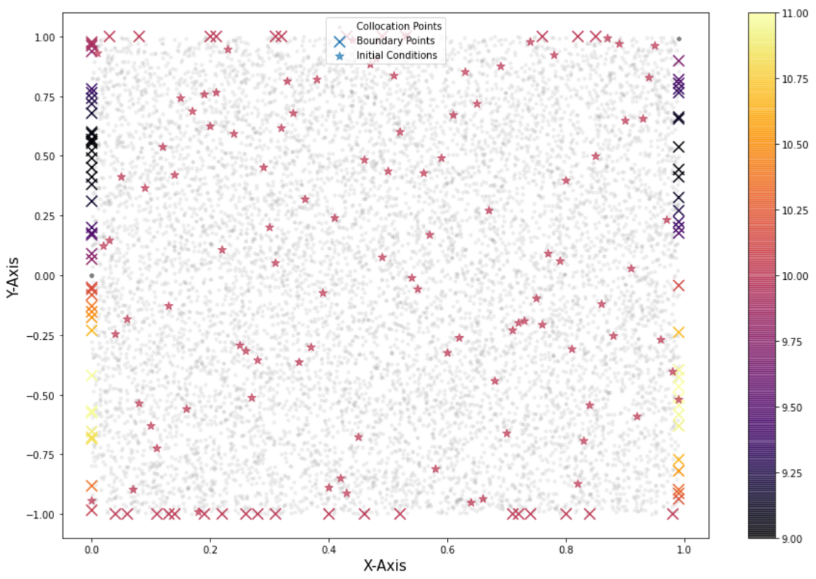


Figure 1: A plot of the training coordinates sampled (N_D, N_{BC} and N_{IC}). This visualizes all the information available to the PINN during training.

3.2 Training

3.2.1 Data Generation

In order for the neural network to be trained to approximate the solution to a PDE within a set of boundaries, it must be trained on points at the boundaries, within the boundaries and at the initial time point. A specified number of points referred to as N_D, N_{BC}, and N_{IC} indicate the number of coordinates used to constrain the solution during training, within the domain, at the boundaries and at the initial state respectively. As can be seen in Figure 1, N_D points were sampled from within the boundaries using a space filling method, Latin hypercube sampling. N_{BC} points were randomly sampled from the boundaries, and N_{IC} points were also sampled randomly from the initial state. This project does not have a conventional dataset, however, these points could be considered as the training data.

3.2.2 Initialising the PINN

When initialising a neural network it is important to avoid the symmetry problem that arises when weights and biases are initialised to be the same value. Proper weight initialisation is necessary to prevent layer activation outputs from diverging to infinity or converging to a local optimum during optimisation. Initially proposed by Glorot and Bengio [10], Xavier initialisation was used to initialise the weights for the input, hidden and output layers of the neural network, whereby values were drawn from a truncated normal distribution centered around zero. Biases were simply initialised to zero, due to the symmetry being broken with the randomisation of the weights.

3.2.3 Activation functions

The activation function that is used in the neural networks is the hyperbolic tangent ($tanh$) function, the same as used by Raissi et al [1]. It is known however, that vanishing gradients is a problem with this activation function. This is due to the exponential function in the gradients preventing smooth gradient flow on the active paths of neurons. This is an area for future research where other activation functions can be used instead, such as the rectified linear unit (ReLU) function. Due to ReLU linearity, it is unlikely to suffer from this short fall and is computationally less demanding as there is no exponential function in its activation [11].

3.2.4 Optimising the PINN

Similar to in the majority of modern deep learning applications, back-propagation is used to adjust the neural network parameters using $Loss_{NN}$ as the cost function. This is done using a stochastic gradient descent algorithm (Adam) along with a quasi-Newton method (L-BFGS) algorithm ([1]). Automatic-differentiation is used to calculate the derivatives (1st and 2nd order when necessary) required for the calculation of the with respect to the input coordinates. The PDE (solved for zero) is explicitly programmed into the loss function, along with the boundary and initial conditions as shown above.

4 Results and Discussion

The PINN was first run on burgers equation to validate the implementation against the solution presented in Raissi et al [1]. This PINN implementation was then applied to the heat diffusion equation in one spatial-dimension with time and two spatial-dimensions. Finally, the PINN was extended further to solve the heat diffusion equation in three dimensions (two spatial and time). This section will outline the results for each scenario along with a comparison of a corresponding numerical method approximation with FEA using the FEniCS library.

4.1 Burgers Equation

The PINN solution for the burgers equation was compared to the numerical matlab solution originally generated in the study by Raissi et al [1]. In addition, the PINN methodology was extended by re-sampling initial and boundary conditions during training. A comparison of both methods is shown below in Figure 2.

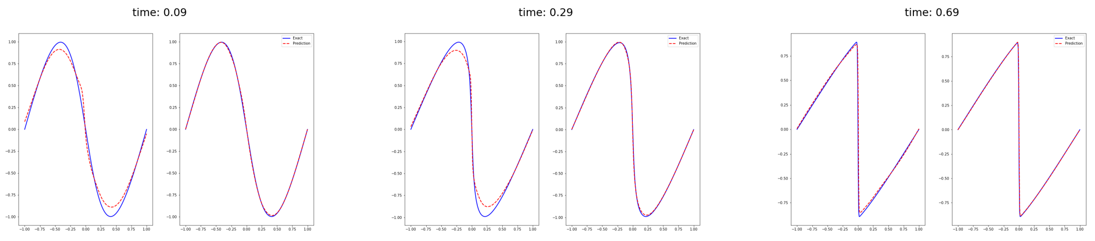


Figure 2: A comparison of the PINN solution using methodology presented by Raissi et al (left) and the extended implementation using re-sampling during training (right) for three time frames.

The PINN implementation was able to successfully replicate the results from Raissi et al, this provided confidence when extending this methodology to other PDE applications. In addition, the training methodology was extended from Raissi et al to include re-sampling of the training coordinates with additional points during the training of the neural network. From 2, its apparent that this addition significantly improved the accuracy and the ability of the PINN to approximate the burgers equation.

4.2 Heat Diffusion in one Spatial-Dimension

One can solve for the heat equation (2) in one spatial dimension by considering a uniform (metal) rod of length L, lying on the x-axis, with a non-uniform temperature distribution. The initial (14) and boundary conditions (15) are thus defined as shown below:

$$IC : u(x,0) = f(x),\ 0 < x < L \tag{14}$$

$$BC : u(0,t) = u(L,t) = 0,\ t > 0 \tag{15}$$

As shown in literature [6], the equation (2) can be dimensionless to make the solution more simpler by grouping physical constants together (16). The thermal diffusivity would become embedded within the time domain, where $T^* = \frac{L^2}{\kappa}$.

$$\hat{x} = \frac{x}{L},\ \hat{t} = \frac{t}{T^*} \tag{16}$$

Assuming the new variables are used, the 1-D dimensionless heat equation can be described as:

$$u_t = u_{xx} \tag{17}$$

$$IC : u(x,0) = f(x),\ 0 < x < 1 \tag{18}$$

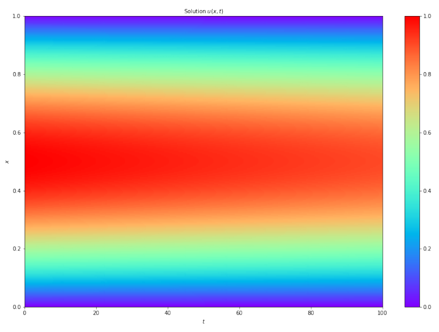
$$BC : u(0,t) = u(1,t) = 0,\ t > 0 \tag{19}$$

Assuming an initial condition of $f(x) = sin\left(\frac{\pi x}{L}\right)$, one can obtain the analytical solution of the 1-D problem (2) as:

$$u(x,t) = sin\left(\frac{\pi x}{L}\right) exp\left(-\kappa\left(\frac{\pi}{L}\right)^2 t\right) \tag{20}$$

It was then assumed dimensionless and a $\kappa = 1cm^2s^{-1}$ ($\approx$ copper) for the later comparison.

The neural network was then trained with a loss function corresponding to the non-dimensional 1D problem (17) with its corresponding conditions, aiming to compare the predicted solution to the analytical one. The results are shown in Figures 3 and 4 for a short time-scale, however, for a more detailed comparison one can look at the individual snapshots, as shown in Figure 5.
Although the PINN gets an accurate approximation to the analytical solution, as the time-scale is increased, a deviation can be seen from the solution. The neural network may be failing to interpret the scale of the decay rate. It could be attempted to solve by feeding the network extra conditional layers to account for this, training the network further, and thus giving the network an additional set of information to the problem.



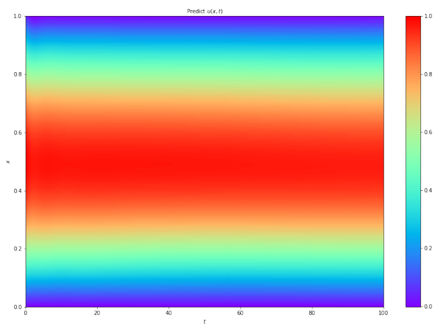


Figure 3: Analytical solution of 1-D heat equation. Figure 4: PINN solution of 1-D heat equation.

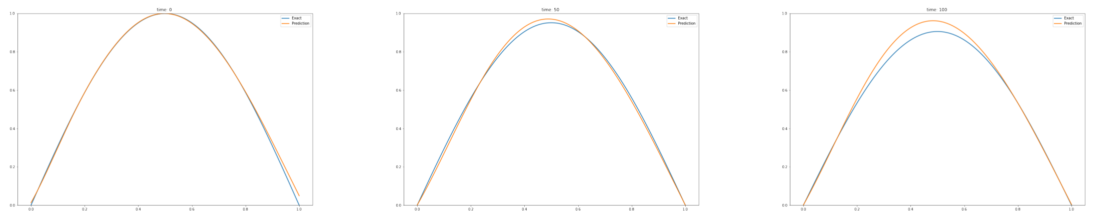


Figure 5: A comparison of the analytical solution to the PINN solution for the 1-D heat equation problem at three different time steps

4.3 Heat Diffusion in two Spatial-Dimensions

The PINN was then applied to solve the heat equation in two spatial-dimensions. This solution was compared to the FEniCS solution, shown in Figures 6 and 7. Specifically, the boundary conditions for this PDE were formulated as follows:

$$BC_{left} = T(0, y) = -sin(y) \tag{21}$$

$$BC_{right} = T(1, y) = sin(y) \tag{22}$$

$$BC_{bottom} = T(x, -1) = 0 \tag{23}$$

$$BC_{top} = T(x, 1) = 0 \tag{24}$$

The PINN solution seems to be able to approximate the heat equation fairly accurately when compared to the FEniCS solution. There is a significant difference in the approximated diffusion of the heat through the material. This is shown by the size of the heat sources and sinks provided at the left and right boundary conditions. However, this difference may be caused by the finite element discretization on its mesh grid, which decreases the granularity of the FEniCS solution. One will

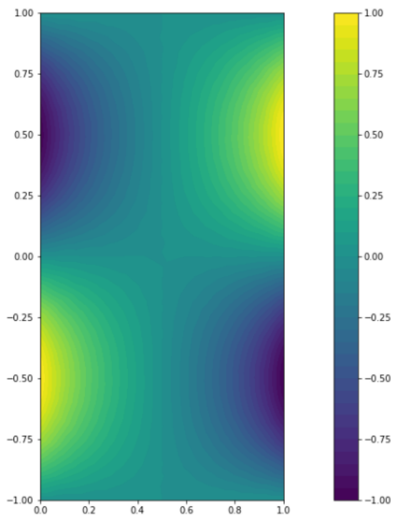


Figure 6: FEniCS solution using the FEA.

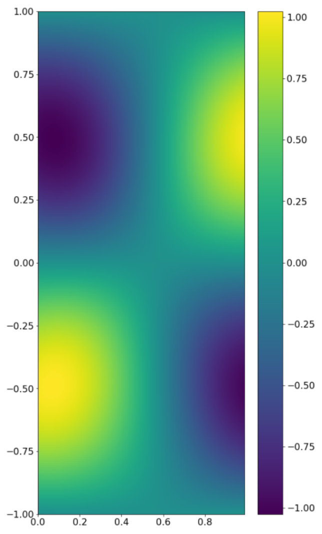


Figure 7: PINN solution.

notice that the resolution of the PINN solution is much higher because its able to provide a continuous solution, where the neural network approximates the solution for any coordinates in the domain. This is a primary advantage of deep learning to approximate solutions to PDEs.

4.4 Time-dependent Heat Diffusion in two Spatial-Dimensions

Lastly, the PINN was extended to solve the heat equation in two spatial-dimensions over time. This was accomplished by formulating the PINN to treat time as a third dimension and by adding initial conditions to the PDE. The boundary conditions remain the same as 21 - 24, while the initial conditions were formulated accordingly:

$$T(x, y, 0) = 10 \tag{25}$$

The approximated solution for the time-dependent heat equation by FEniCS and the PINN can be seen in Figures 8.

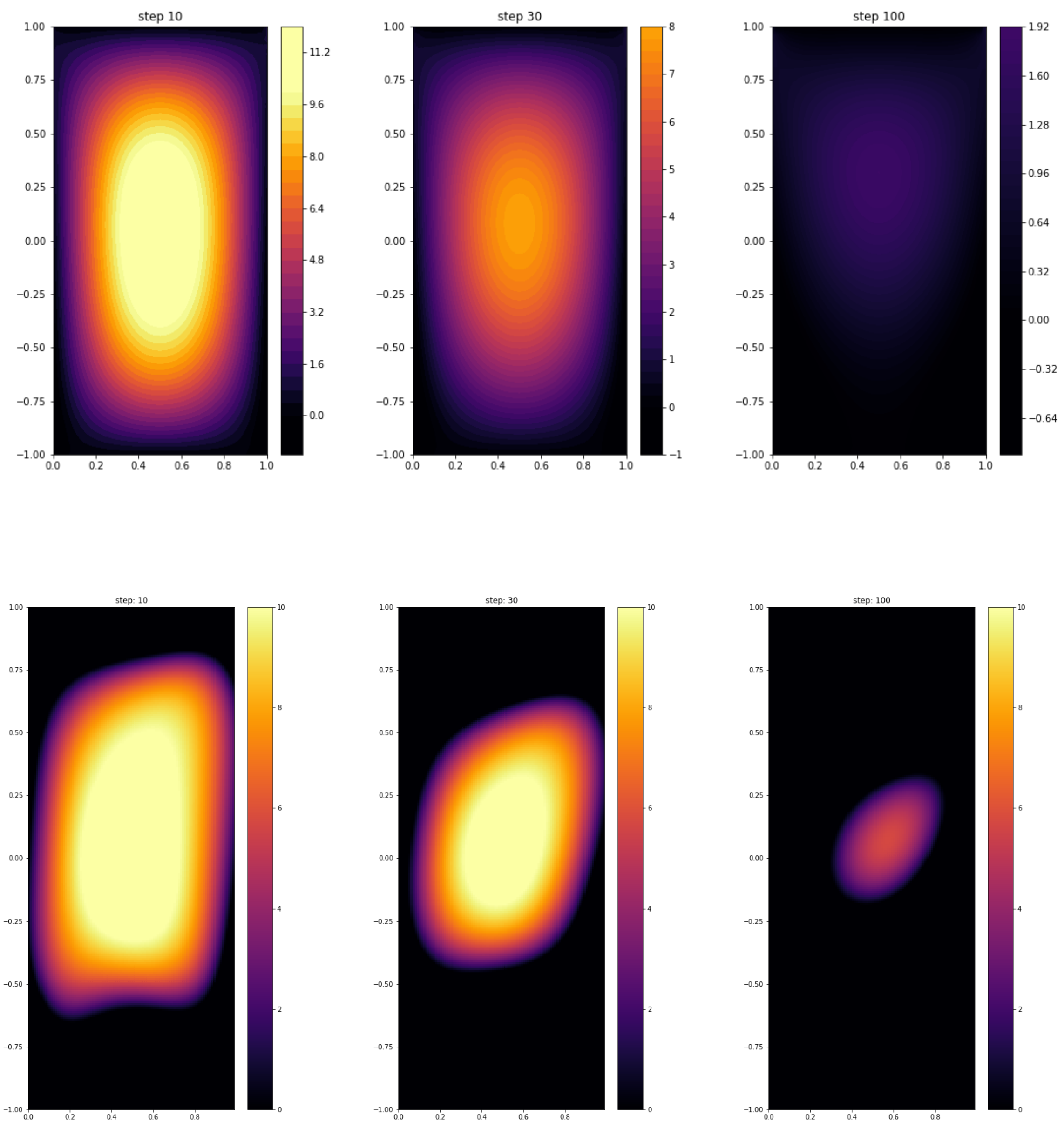


Figure 8: Comparison of the FEniCS FEA solution (top) for heat equation with two spatial-dimensions with the PINN solution at three time frames in time.

One will notice that the PINN approximated solution is fairly close to the FEA solution. However, there are apparent differences, the PINN seems to converge upon a more drastic change in temperature over some regions. In addition, the PINN exhibits more variability and randomness in its solution, which may more closely reflect reality, but may be less accurate when compared to the theoretical solution. Lastly, the PINN solution performs surprisingly well in estimating the rate of temperature decay from the uniform starting initial temperature, provided it is only given the coordinates for training.

5 Conclusions

As demonstrated, utilizing modern deep learning techniques for solving PDEs could allow for significant improvements in physics modelling. Advances in this area have the potential to positively impact countless industries. Two of the most important physics phenomenon affecting industries are fluid flow and heat transfer, which can be described by burgers equation and the heat equation respectively. Hence, these equations were chosen as the primary topics of study regarding application of PINNs. Specifically this project involved validation the PINN implementation with the burgers equation. The PINN was then applied to the heat equation in two spatial-dimensions. Finally, the PINN was extended to a higher dimension by approximating a solution to the time-dependent heat equation in two spatial-dimensions. The remainder of this section will summarize the main findings as well as explore further research in the area.

5.1 Main Findings

Ultimately, the PINN solution was able to accurately approximate the solution for the burgers equation. The accuracy seemed to diminish when applying the same methodology to a different PDE, the heat equation. Some of this discrepancy could be explained by the discretization of FEA compared to the continuous higher resolution solution of the neural network. In addition, it may be necessary to tune hyper-parameters including neural network architecture, learning rate, and tolerance for each PDE application.

5.2 Further Research

By expanding our research into the three dimensional space of PDEs we open up a world of potential further research. Climate change and rising sea levels pose an eminent threat to mankind, by using PINNs we can use deep learning to approximate ocean currents and the potential flood zones along the coast. This will allow for neural network informed government policy. PINNs are not as computationally demanding compared to other methods, allowing for faster approximations and save large amounts of energy.

There still seems to be some discrepancies between our solution and other conventional methods such as those shown using FEniCS. We believe that this is due to the network being unable to approximate the decay rate as mentioned in main findings. Future research could try understand why these differences occur, potentially through the use of other activation functions, additional data generation to be used as input or varying the structure of the network.

References

[1] 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.

[2] Zhuming Bi. *Finite Element Analysis Applications*. Academic Press, 2019.

[3] Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115(34):8505–8510, 2018.

[4] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998.

[5] Maziar Raissi and George E. Karniadakis. Hidden physics models: Machine learning of nonlinear partial differential equations. *CoRR*, abs/1708.00588, 2017.

[6] Hancock and Matthew Hancock. 18.303 linear partial differential equations, fall 2004, 01 2004.

[7] C Basdevant, M Deville, P Haldenwang, J.M Lacroix, J Ouazzani, R Peyret, P Orlandi, and A.T Patera. Spectral and finite difference solutions of the burgers equation. *Computers Fluids*, 14(1):23–41, 1986.

[8] Mats G. Larson and Fredrik Bengzon. *The Finite Element Method: Theory, Implementation, and Applications*. Springer Berlin, 2015.

[9] Robert C. Kirby. Algorithm 839: Fiat, a new paradigm for computing finite element basis functions. *ACM Transactions on Mathematical Software*, 30(4):502–516, 2004.

[10] Xavier Glorot. Understanding the difficulty of training deep feedforward neural networks. *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks, Jun 2011.