Solving Burger Equation with Multiple Dimensions by Using Physics-Informed Neural Networks and Improving Models

Supervisor: Anirbit *anirbit.mukherjee@manchester.ac.uk*
Department of Computer Science
The University of Manchester

Student: Zhihao Fang *zhihao.fang@manchester.student.ac.uk*
Student ID: 10732778
Department of Computer Science
The University of Manchester

Contents

List of Figures

List of Tables

Abstract

For a long time, people have struggled to solve numerical partial differential equations (PDEs), particularly in high-dimensional settings. In recent years, with the proposal of "AI for science", scientists have found another way to solve PDE — using deep learning. To solve partial differential equations effectively and accurately, I will combine the Physics-Informed Neural Network(PINN) method with deep learning for different models. The main goal is to improve the ability of the model to predict. This project explores the use of neural networks to solve PDEs, with a focus on the Burger Equation. It will contain one-dimensional, two-dimensional, and three-dimensional Burger Equation. At the same time, the phenomenon of Double Decent and the inverse problem of the Burger Equation will also be briefly discussed. The document aims to provide insight into current studies on neural network solutions for PDEs and offers potential avenues for future research in this area.

Keywords: PDE, PINN, Burger Equation, Improving model

Acknowledgements

I am deeply grateful to my supervisor, Dr.Anirbit.Mukherjee, for his invaluable guidance, support, and encouragement throughout the duration of this project. His expertise in Deep learning has been instrumental in shaping my understanding of the subject matter and providing me with the necessary resources to complete this work.

Dr.Anirbit.Mukherjee's constructive feedback, unwavering belief in my abilities, and commitment to my growth as a researcher have served as constant sources of motivation and inspiration. I am truly appreciative of the time and effort he has devoted to my academic development.

In addition, I would like to extend my thanks to the Department of Computer Science of The University of Manchester for providing me with a stimulating and nurturing academic environment. The knowledge and skills I have gained during my time here will undoubtedly prove valuable as I continue on my academic journey.

1 Introduction

1.1 Context

"AI for Science" is listed as an important trend, and it is no doubt to see the huge potential brought by the combination of artificial intelligence and traditional scientific research. And now more researchers start to study this and accelerate the progress of this scientific revolution.

Today, artificial intelligence is almost completely integrated into our lives, and it is almost everywhere. Therefore, it is becoming more and more popular to use artificial intelligence to solve basic science, and solving partial differential(PDE) problems is a very important part of it. PDE can be difficult to solve, however, neural networks have shown to be an effective tool in this area. This is because they are able to recognize intricate, non-linear correlations between input and output data, which frequently characterize the behavior of a physical system controlled by PDEs.

The usage of physics-informed neural networks(PINN) is one method for applying neural networks to the solution of PDEs. (Cuomo et al., 2022) and (Ray et al., 2023) consider that both actual data and well-known physical laws governing the system under study are used to train these networks. The network architecture of PINNs can more effectively forecast the behavior of the system under study and offer stronger solutions to PDEs by including these physics constraints.

1.2 Motivation

While physics-informed neural networks (PINNs) have shown great promise in solving partial differential equations (PDEs), they are not always guaranteed to provide a better solution.
From (Krishnapriyan et al., 2021) and (Wang et al., 2021) a type of model called Physics-Informed Neural Networks (PINNs) has been discovered through scientific machine learning research. The basic strategy is to use existing machine learning techniques to train the model and incorporate physical domain information as soft constraints into an empirical loss function. Currently, despite the fact that current PINNs techniques can easily fail to learn significant physical phenomena for even somewhat difficult problems, they can still learn effective models for comparatively simple situations.
After examining the fundamental principles of PINN, it becomes apparent that the model fails to incorporate information from boundary or initial conditions, rendering existing knowledge unusable. Consequently, this project aims to explore a viable method for enhancing the accuracy of the Burger Equation. Furthermore, this endeavor has sparked my curiosity regarding the scientific applications of machine learning. Identifying solutions through AI approaches can significantly foster interdisciplinary linkages and augment computational intelligence in a circular fashion since PDEs have widespread usage across various domains such as physics, finance, and engineering.
I seek out challenging issues because I am passionate about artificial intelligence, particularly in the area of partial differential equations. I'm really at ease exploring and resolving these challenging issues utilizing my understanding of mathematics and experience with artificial intelligence. We can now easily address some of the most challenging problems in this area thanks to the new and inventive ideas brought about by the development of artificial intelligence.

1.3 Current studies

Many scientific and engineering topics are studied using partial differential equations (PDEs). For the solution of partial differential equations using traditional numerical techniques, such as finite difference methods (Thomas, 2013) or finite element methods(Felippa, 2004). However, because we must perform several computations and, of course, there are instances when we are unable to utilize conventional numerical methods to obtain the precise solution, it is not always appropriate for difficult issues or high-dimensional spaces.
Therefore, given the understanding of deep learning, several scientists have attempted in recent years to use neural networks to solve partial differential equations. Instead of using conventional numerical methods to find precise answers, the concept is to employ neural networks to produce approximations of partial differential equation solutions.
From several viewpoints, a number of methods have been developed, including the deep Ritz method (DRM)(Yu et al., 2018), deep Galerkin method (DGM)(Sirignano & Spiliopoulos, 2018), generative adversarial networks (GAN)(Creswell et al., 2018) and physically informed neural networks(PINN)(Wang et al.,

2021).
For the deep Ritz method, it uses a trial solution that satisfies the boundary conditions and comes close to the true PDE solution and then employs a neural network to learn the correction term that causes the trial solution to satisfy the PDE precisely. Another method that approximates a PDE solution using neural networks is the deep Galerkin method. In this approach, the solution itself is not used. This method does not make use of the solution itself. Thus, we can get the differential operator in the PDE by using the approximation by a neural network. The method can be used to solve different PDEs as long as general trial solutions are not necessary. By figuring out the distribution of possible solutions for a particular PDE, generative adversarial networks (GANs) can also be used to solve PDEs. After learning the distribution, new solutions can be produced by sampling from it. The last class of neural networks is known as physically informed neural networks (PINNs), which are created specifically to include known physical constraints during the training process. The PDE itself, as well as any boundary conditions or additional physical laws that are relevant to the current issue, can all be considered as constraints. PINNs can learn to approximate the solution to a PDE and in the meanwhile satisfy any known physical constraints by including these constraints such as boundary-included and initial-included in the loss function, the neural network will minimize the loss function by using optimization methods such as gradient descent(Raissi et al., 2019b).

In the previous, we find several ways to forward solve partial differential equations. However, exploring the inverse problem(Raissi et al., 2019b) in PDEs is also very important. Because it allows us to determine the unknown parameters or initial/boundary conditions of complex systems, and we need to give the observed solution. We can give an example in medical imaging, in a common situation, the doctor may have an image of an internal organ but lack information about its internal structure or composition. By solving the inverse problem, we could use the image data to determine the missing information and make more accurate diagnoses.
By reading the paper (Meng & Karniadakis, 2020), I know that exploring the inverse problem in PDEs is a much more challenging but important area of research that has wide-ranging applications and can lead to significant advancements in many fields.

1.4 Aims and Objectives

After consulting relevant information (Fangohr, 2016) and (Chen et al., 2020), I found that the model can be changed through boundary conditions and initial conditions. In my opinion, two methods that can to enhance the efficiency of physics-informed neural networks (PINNs) in solving partial differential equations are boundary-included models and initial-included models for our PDEs.
Boundary-included models include integrating the network design with a PDE's known boundary conditions. This will enable the network to learn the underlying physics more precisely and generate more precise solutions. Traditional methods may not be as successful in solving issues with clearly defined boundary conditions or boundary layers.
On the other hand, initial-included models build the network architecture using the PDE's initial conditions. This can be especially useful for time-dependent issues where the beginning conditions and underlying PDE both influence the system's behaviour. The network can better forecast how the system will behave over time by incorporating this information into its architecture.
We hope to find that at least one of these methods can help PINNs be more accurate and resilient when solving PDEs. In this work, I choose Burger equation, The Burger equation is a good candidate for research on PINNs because it is a nontrivial, nonlinear PDE that can be difficult to solve using traditional numerical methods. Our experiments involve solving Burger equation for one-dimensional, two-dimensional, and three-dimensional cases, and by comparing our two proposed models, we determine that integrating initial conditions into a common PINN can significantly improve performance in all dimensional cases.

1.5 Report Structure

The project report is divided into the following sections:
1. Introduction: This section provides current studies of how to use neural network solving PDE, my motivation, and the objective of this project.
2. Background: In this part, I discuss the origin of PDEs. Then, I will list some common PDE equations, and then lead to the PDE equation that needs to be used in this report: Burger Equation.
3. Methodology: In this part, I will briefly explain the principles of deep learning and a framework of

deep learning that I use: DNN. On the other side, PINN involves the use of scientific computing devices in conventional numerical domains, particularly to address diverse issues involving partial differential equations. What's more, I will introduce my fractional error and SGD algorithm. Finally, briefly talk about Double Decent.
4. Experiments and Analysis: My experiment is roughly divided into four parts, the first two are Inviscid Burger Equation and Viscid Burger Equation, in order to explore whether different models can improve the accuracy of the predicted value. The third part is the Inverse Problem for Burger Equation, and the fourth part(Double Decent) is to observe whether the Burger Equation will appear as Double Decent phenomenon.
5. Summary and Outlook: This section summarises the report's main findings. There are recommendations for future research in the field of neural networks for solving partial differential equations, as well as potential future developments.

2 Background

2.1 The origin of partial differential equations

From (Renardy & Rogers, 2006), An ordinary differential equation (ODE)(Hartman, 2002) is a differential equation with only one independent variable and an unknown function as a differential equation for short; If a function of more than one variable has partial derivatives in a differential equation, or if an unknown function is connected to and has derivatives with regard to more than one variable, the equation is a partial differential equation.

Due to the quick advancement of science and technology, it is now important to adequately express a number of investigated issues by using functions of many variables. Physical quantities, for instance, have a variety of features; for instance, states of tension characterising an item someplace are known as tensors; temperature, density, and other similar attributes are expressed in numerical terms and are known as scalars; and so on. Velocities, electric field attractive forces, and other variables vary in both magnitude and direction. These values, which are represented by functions of many variables, relate not only to time but also to spatial coordinates.

In the seventeenth century, the field of calculus equations was established. The special partial differential equation was initially put out by the French mathematician D'Alembert in his book "On Dynamics" shortly after Euler first introduced the second-order equation of string vibration in his work. At the time, hardly much attention was paid to these works. D'Alembert attempted to demonstrate that other curves other than sinusoids constitute modes of vibration in his work "A Study of the Curves Formed in the Vibration of Strings under Tension" in 1746. In this way, the study of string vibrations gave rise to the field of partial differential equations.

Contemporaneously with Euler, the Swiss mathematician Daniel Bernoulli investigated issues in mathematical physics and put out a broad approach for comprehending the vibration of elastic systems, which had a significant influence on the creation of partial differential equations. The concept is further developed by Lagrange's discussion of first-order partial differential equations.

2.2 Laplace equation

Potential equations played a key role in the 19th century's advancement of partial differential equations. The representative figure in this regard was G. Green, a self-taught British mathematician who was born as a miller. The potential equation is also called the Laplace equation(Medková, 2018):

$$\Delta \mathrm{V} = \frac{\partial^2 \mathrm{V}}{\partial x^2} + \frac{\partial^2 \mathrm{V}}{\partial y^2} + \frac{\partial^2 \mathrm{V}}{\partial z^2} = 0$$

Green is the founding father of the Cambridge School of Mathematical Physics. His work has cultivated powerful successors such as W.Thomson, G.Stokes, and J.C.Maxwell. They are a typical nineteenth-century mathematical physicist. Their main goal is to develop general mathematical methods for solving important physical problems, and their main weapon is partial differential equations so that in the nineteenth century, partial differential equations have almost become synonymous with mathematical physics.

The contributions of the Cambridge School of Mathematical Physics revived British mathematics in the nineteenth century after more than a century of silence. The electromagnetic field equation derived by

Maxwell in 1864

$$\begin{aligned}\operatorname{rot} H &= \tfrac{1}{c}\tfrac{\partial(\varepsilon E)}{\partial t},\\ \operatorname{rot} E &= -\tfrac{1}{c}\tfrac{\partial(\mu H)}{\partial t},\\ \operatorname{div}(\varepsilon E) &= \rho,\\ \operatorname{div}(\mu H) &= 0\end{aligned}$$

It was the most dramatic victory in mathematical physics in the nineteenth century. In addition to having a significant influence on science and business, Maxwell's theories on electromagnetic waves also contributed to the fame of the partial differential equations he discovered while studying this particular set of equations. In his address honoring Maxwell, Einstein remarked that "Partial differential equations entered theoretical physics as handmaidens, and then gradually became housewives." had a significant impact on this transition.

2.3 Navier–Stokes equations

The Navier-Stokes equation was independently created by Irish mathematician Georges Gabriel Stokes and French mathematician Claude-Louis Navier in the 19th century.
The equations for viscous flow that are now known as Navier's equations were included in Navier's first publication of his fluid mechanics work in 1827. In a paper written in 1845, Stokes came up with an equation for viscous flow. Stokes did, however, take into account the Stokes flow scenario.
The Navier-Stokes equations, which describe the motion of fluids in space and time, were created in 1857 by combining the Navier and Stokes equations. These equations, which are fundamental to comprehending fluid dynamics, are based on the ideas of mass, momentum, and energy conservation.
The Navier-Stokes equation, which is the fundamental equation in hydrodynamics, explains the motion of the fluid (Schneiderbauer & Krieger, 2013). And from (Wang, 1991), the constant-property Navier–Stokes equations can be written as:

$$\rho\left(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u}\cdot\nabla)\mathbf{u}\right) = -\nabla p + \eta\nabla^2\mathbf{u} + \mathbf{f}$$

where the terms:
ρ - Fluid density [Units: $\mathbf{kg}\cdot\mathbf{m}^{-3}$]
$\mathbf{u}$ - Fluid flow velocity [Units: $\mathbf{m}\cdot\mathbf{s}^{-1}$]
∇ - Gradient operator, $\nabla = \partial_x\mathbf{i} + \partial_y\mathbf{j} + \partial_z\mathbf{k}$
p - Fluid pressure [Units: $\mathbf{Pa}$]
η - Dynamic viscosity [Units: $\mathbf{Pa}\cdot\mathbf{s}$]
∇^2 - Laplacian operator, $\nabla^2\mathbf{u} = \partial^2_{xx}\mathbf{u} + \partial^2_{yy}\mathbf{u} + \partial^2_{zz}\mathbf{u}$
f - Body forces per unit volume (e.g. gravity per unit volume) [Units: $\mathbf{N}\cdot\mathbf{m}^{-3}$]

Several disciplines, including engineering, physics, meteorology, oceanography, and aerodynamics have been significantly impacted by the Navier-Stokes equations. They are used to model blood flow in the human body, construct airplanes, examine weather patterns, and investigate ocean currents.
The Navier-Stokes equations, despite their significance, are notoriously challenging to solve analytically for the majority of real applications due to their intrinsic nonlinearity and complexity. In order to approximate the solutions to these equations, researchers have created numerical methods and computational approaches, which have resulted in numerous significant advancements and applications in fluid mechanics.

2.4 Burger equation

The Burger equation(Mittal & Singhal, 1993) is a non-linear partial differential equation that was first derived by John Martin Burger in 1948. It was originally used to describe turbulence in fluid mechanics but has since been applied to a wide range of physical, biological, and economic systems. The equation has a simple form, which makes it a popular choice for illustrating various concepts in mathematical physics, including shock waves and solitons. Today, the Burger equation continues to be an active area of research in many fields, with new applications and solutions being discovered all the time.
The 1D form above is the most commonly recognized Burger equation, and it can be written as:

$$u_t + uu_x = \nu u_{xx}$$

where u is a function of time t and space x, and ν is a constant known as the viscosity coefficient. This formula represents how quickly a fluid or wave flows across space and time.
The non-linear term uu_x represents the convective transport of the fluid, while the linear term $\nu u_x x$ represents the diffusive spreading of the fluid. The system's overall behavior, as defined by the equations, depends on how these two factors are balanced.
One of the key features of the Burger equation is that it can produce shock waves, which are abrupt changes in fluid velocity. These shock waves are an important aspect of the equation, and they have been used to model a variety of physical phenomena, including traffic flow, sonic booms, and waves in the ocean.
There are several numerical methods that can be used to solve the Burger equation, including finite difference methods, spectral methods, and finite volume methods. In addition, analytical solutions have been derived for specific cases of the equation, which provide insight into its behavior.
Overall, the Burger equation is a simple but powerful tool for studying the behavior of non-linear systems. Its continued relevance and importance in a wide range of fields attest to its versatility and value as a model for complex processes.

A partial differential equation that describes the dynamics of fluid flow is the Berger equation. It is a condensed form of the Navier-Stokes equation and is often employed in a number of disciplines, including statistical physics, nonlinear dynamics, and fluid mechanics. A partial differential equation is the Burger equation. Here, it is possible to express it in a generic dimensional form, and both the boundary conditions and the beginning conditions are provided:

$$\begin{gathered}\partial_t \boldsymbol{u} + (\boldsymbol{u} \cdot \nabla)\boldsymbol{u} = \nu \nabla^2 \boldsymbol{u}, (\boldsymbol{x}, t) \in [0,1]^d \times [0,1] \\ \boldsymbol{u}(\boldsymbol{x}, 0) = F(\boldsymbol{x}), \boldsymbol{x} \in [0,1]^d \\ \boldsymbol{u}(\boldsymbol{x}, t) = \boldsymbol{g}(\boldsymbol{x}, t), \boldsymbol{x} \in \partial([0,1]^d)\end{gathered}$$

Then, we can get the matrix for every part in the formula:

$$\partial_t \begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_d \end{bmatrix} = \begin{bmatrix} \partial_t u_1 \\ \partial_t u_2 \\ . \\ . \\ . \\ \partial_t u_d \end{bmatrix}, \ (\sum_{i=1}^{d} u_i \partial_{x_i}) \begin{bmatrix} u_1 \\ u_2 \\ . \\ . \\ . \\ u_d \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^d u_i \partial_{x_i} u_1 \\ \sum_{i=1}^d u_i \partial_{x_i} u_2 \\ . \\ . \\ . \\ \sum_{i=1}^d u_i \partial_{x_i} u_d \end{bmatrix}, \ \nu \begin{bmatrix} \nabla^2 u_1 \\ \nabla^2 u_2 \\ . \\ . \\ . \\ \nabla^2 u_d \end{bmatrix} = \nu \begin{bmatrix} \sum_{i=1}^d \partial_{x_i}^2 u_1 \\ \sum_{i=1}^d \partial_{x_i}^2 u_2 \\ . \\ . \\ . \\ \sum_{i=1}^d \partial_{x_i}^2 u_d \end{bmatrix}$$

The given expressions represent the partial differential equations of a system of d scalar fields u_1,u_2......u_d. The first equation represents the time evolution of each field, the second equation represents the advection of each field along its gradient, and the third equation represents the diffusion of each field with a diffusion coefficient ν. The Laplacian operator ∇^2 is the sum of the second derivatives of each field with respect to their respective spatial dimensions.

2.5 Other famous PDEs

In addition to Maxwell's equations(Huray, 2011), the famous partial differential equations derived in the nineteenth century include C.L.M.H.Navier-Stokes for viscous fluid motion and Cauchy's equation for elastic media.
For the many types of differential equations established in the 18th and 19th centuries, the efforts of mathematicians to find explicit solutions often failed, which prompted them to turn to prove the existence of solutions. Cauchy was the first mathematician to consider the existence of solutions to differential equations(Simpson, 1984).

There are frequently highly generic approaches that are initially not precisely mathematically justified and applied to a wide variety of physical issues while investigating the specific differential equations that occur in the solution of physical problems. These include, for instance, the Fourier technique, Ritz method, Galerkin method, perturbation theory method, etc. One of the justifications for trying to carefully prove these methodologies' applicability is their validity. As a result, new mathematical theories and study focuses (such as the Fourier integral theory, the intrinsic function expansion theory, and the generalized function theory, etc.) are developed.

3 Methodology

3.1 Deep learning

A subset of the larger artificial intelligence technique family that enables computers to imitate human behavior is machine learning. With the use of these methods, an algorithm can be taught to carry out specific tasks without being explicitly programmed.
Machine learning is a subset of deep learning(LeCun et al., 2015). In contrast to machine learning, these algorithms automatically extract relevant patterns from unprocessed data and utilize those patterns as features to train the system to do a certain task. This technique is comparable to how the human brain works. Its core consists of neural networks, algorithms that were motivated by the nature of connections between neurons in our brains, hence the name.

In recent years, the computational and applied mathematics communities have been very interested in deep neural networks (DNNs) because of their capacity to resolve high-dimensional partial differential equations. See (Weinan et al., 2021), (Wang et al., 2021) and (Beck et al., 2020) for reviews and references here. So I try to use this in my experiment.

3.1.1 Perceptron

Frank Rosenblatt created the perceptron artificial neural network in 1957 when he was employed at the Cornell Aeronautical Laboratory. When reduced to its most basic form, a binary linear classifier, it may be thought of as a feed-forward neural network.
Perceptron has four parts: input and output; weight and bias; Net sum; activation function. As shown below:

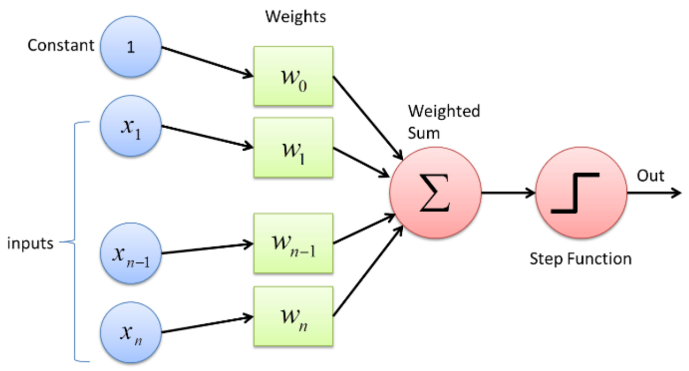


Figure 1: Single perception

The final output of the node above, we can call it $\hat{y}$ and it is produced by the perceptron by taking this single value and processing it through a non-linear activation function. Here this is the forward propagation process. A simplified illustration is presented in Figure 1. And it can be expressed in terms of linear algebra as:

$$\hat{y} = g\left(w_0 + X^T W\right)$$

Where: $X = \begin{bmatrix} x_1 \\ \vdots \\ x_{n-1} \\ x_n \end{bmatrix}, W = \begin{bmatrix} w_1 \\ \vdots \\ w_{n-1} \\ w_n \end{bmatrix}$ and w_0 is the bias term accompanying the input values and most of the time is 1.

3.1.2 Activation function

The dynamics of deep network training and task performance can be greatly impacted by the choice of activation functions. Among the most popular and effective activation functions used today, the rectified linear unit (ReLU) is widely regarded as the best option. However, the (Ramachandran et al., 2017), show that the best-discovered activation function, the sigmoid activation function frequently outperforms ReLU on deeper models across a variety of difficult datasets. So sigmoid activation function will be used in the experiments. The plot is below:

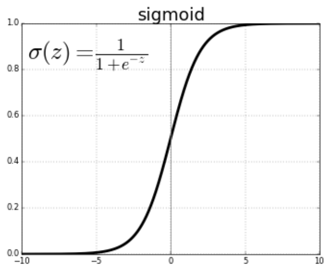


Figure 2: Activation function for Neural network

3.1.3 Affine Function

Affine functions are functions composed of polynomials of order 1, and the general form is:

$$f(x) = Ax + b$$

Among the formula above, A is an m × k matrix, x is a k vector, b is an m vector, which actually reflects a transformation from k The role of the affine function is to change the dimension or change the shape and direction of the spatial mapping relationship from dimension to m dimension. This process is called an affine transformation.
The calculation is carried out via an affine transformation in each fully connected layer of the neural network. To accomplish the required transformation, the weight and bias parameters can be optimized using gradient descent-related algorithms like Adam (Kingma & Ba, 2014). The objective of this optimization procedure is to use the optimized weights and biases to get the result of the affine transformation as near as feasible to the real value.

3.1.4 Multiple-layer Perception(MLP)

From the (Taud & Mas, 2018), We can see that the major issue with a single-layer perception is that it fails to account for the dataset's non-linearity and, as a result, doesn't perform well on non-linear data. Multi-layer perception, which excels on non-linear datasets, may easily fix this issue.
Multi-layer perception can have several hidden layers in between the input and output levels. As an example, the MLP depicted in the figure 3 below, there are three inputs: one output y, two hidden layers of five neurons each, and two hidden layers. The following layer's neurons are entirely linked to all of the neurons in the prior layer, which is referred to as full connectivity.

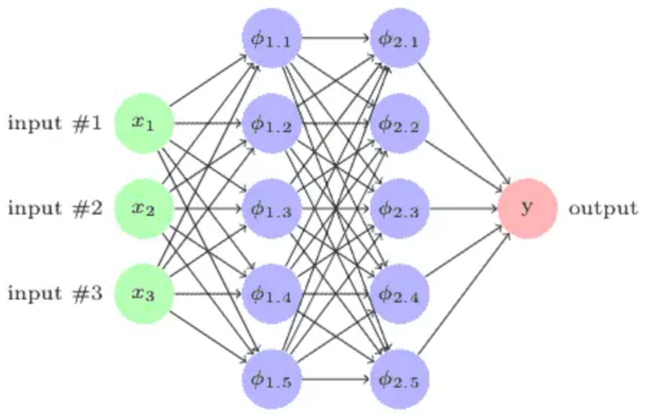


Figure 3: An example for MLP

3.2 PINN

PINN (Physics-Informed Neural Network) is a type of neural network that incorporates physical laws and constraints into its training process. This is achieved by adding additional terms to the loss function that enforce these constraints. In this report, I will try to use the residual as a loss function in a PINN.

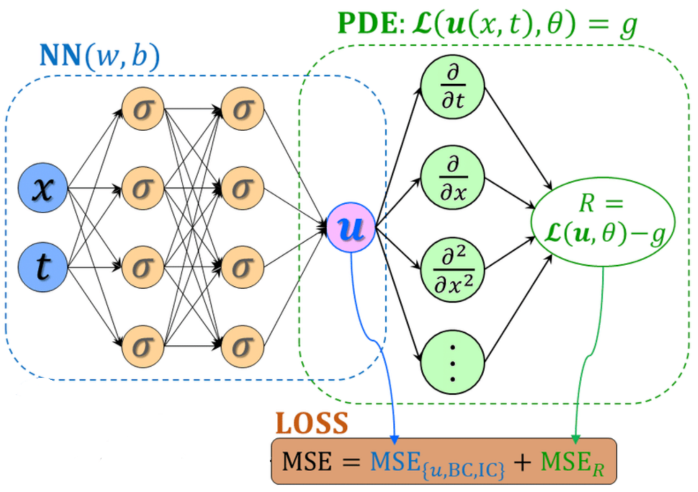

PDE: $\mathcal{L}(u(x,t),\theta) = g$
NN(w,b)
x
t
σ
u
$\frac{\partial}{\partial t}$
$\frac{\partial}{\partial x}$
$\frac{\partial^2}{\partial x^2}$
$\vdots$
$R = \mathcal{L}(u,\theta) - g$
LOSS
MSE = MSE$_{\{u,\mathrm{BC},\mathrm{IC}\}}$ + MSE$_R$

Figure 4: PINN structure

Figure 4 is the basic structure of PINN. The physical equation "participated" in the training process by modifying the loss function of the neural network by the physical equation's difference between before and after each iteration. So that the final training result meets the physical law, the neural network optimizes both the network's own loss function and the difference between each iteration of the physical equation throughout the training phase. Let's analyze the loss function. If the neural network can solve the PDE solution well, then for any point from the initial or boundary value, its value MSE_u tends to zero; for the internal configuration points, because the differential equation is well fitted, MSE_R will also tends to Zero, that is to say, when the value of the loss function is 0, we can say that the value of the neural network tends to the true value at each point on the training set. In this way, the problem is transformed into how to optimize the loss function.

3.2.1 Problem setup

Below, we present the problem scenario in order to demonstrate the general form of PDEs that PINN can solve.
Think about a nonlinear, parametrized PDE with d spatial dimensions:

$$\frac{\partial u(x,t)}{\partial t} + \mathcal{L}[u(x,t);\lambda] = 0, (x,t) \in \Omega \times [0,T]$$

where $u(x,t)$ denotes the latent (hidden) solution, $\mathcal{L}[\cdot;\lambda]$ is a nonlinear operator parametrized by λ , and Ω is a subset of $\mathbb{R}^d$.
Conservation rules, diffusion processes, and advective-diffusion reaction systems are only a few examples of mathematical physics that are covered by the fundamental forms offered for a variety of partial differential equations.

3.2.2 Neural network

A PINN model typically consists of an input layer, one or more hidden layers, and an output layer. The input layer takes in the input variables, which could include parameters such as spatial coordinates or time. The hidden layers use nonlinear transformations to map the inputs to a higher-dimensional space, while the output layer produces the predicted output.
In addition to the standard neural network architecture, a PINN model also includes additional terms in

the loss function that enforce physical constraints. These constraints are typically expressed as differential equations or boundary conditions.
In general, the Neural network $N : \mathbb{R}^d \to \mathbb{R}^n$, where d is the number of dimensions and n is the number of output.

3.2.3 Model with improvement

The resulting PINN model is able to learn the underlying physics of a system, along with the corresponding boundary and initial conditions, from observational data. This makes it a powerful tool for modeling complex physical systems that may be difficult to analyze using traditional analytical methods.

The three different types of models - vanilla, boundary-included, and initial-included - are used in the context of solving partial differential equations (PDEs) using PINN's neural networks. The domain for $\vec{x}$ is $[a, b]$ and the domain for t is [0,T].

The vanilla model refers to a standard neural network architecture that is trained solely on observational data, without incorporating any additional physical constraints or boundary/initial conditions. This approach can be useful in situations where the underlying physics of the system is well-understood, and the focus is mainly on accurately fitting the data.

$$\text{model}_{\text{Vanilla}}(\boldsymbol{x}, t) := \text{Net}(\boldsymbol{x}, t)$$

With the exception of the observation sites, boundary-included models essentially make known boundary conditions training-free by including them in the training process. When boundary conditions are crucial in establishing how the system behaves, this method is extremely helpful.

$$\text{model}_{\text{boundary-included}}(\boldsymbol{x}, t) := \text{Net}(\boldsymbol{x}, \text{t}) \cdot \frac{-a + \boldsymbol{x}}{b - a} \cdot \frac{b - \boldsymbol{x}}{b - a} + \frac{b - \vec{x}}{b - a} \cdot g_a(a, t) + \frac{-a + \boldsymbol{x}}{b - a} \cdot g_b(b, t)$$

Similar to training-free solutions, initial inclusion models successfully anticipate initial circumstances by incorporating known initial conditions into the training process in addition to observation points. This method is especially useful in situations where anticipating the beginning circumstances accurately is essential to predicting how a system will evolve over time.

$$\text{model}_{\text{initial-included}}(\boldsymbol{x}, t) := \text{Net}(\boldsymbol{x}, \text{t}) \cdot \frac{t}{T} + \frac{(T - t)}{T} \cdot u_0(\boldsymbol{x}, 0)$$

In general, the choice of which type of model to use depends on the specific problem being solved, and the level of understanding of the underlying physical constraints of the system. Incorporating physical constraints can help to ensure that the resulting predictions are physically meaningful and consistent with the observed behavior of the system.

3.2.4 Empirical risk with improvement

Using the residual as a loss function in a PINN involves computing the difference between the predicted outputs of the network and the true values, and then minimizing this difference. By minimizing the residual loss, the PINN is able to learn the governing equations of a physical system, along with the corresponding boundary and initial conditions. From (Sirignano & Spiliopoulos, 2018), we can construct the objective function for the population risk form. But what needs to be considered in our experiment is the empirical risk, so we need to rewrite the population risk from the paper.

Here we can show the vanilla PINN approach(for boundary-included and initial-included is almost the same), using the mean square error as the empirical risk of the approximator. We shall define model $f(t, x; \theta)$ as the approximator where θ is the parameters in the neural network. What's more, let $\mathcal{D}$ be the points set sampled from the domain of the PDE. The empirical loss $\hat{\mathcal{R}}$ of our equation is given by:

$$\hat{\mathcal{R}}_{\text{vanilla}} = \frac{1}{|\mathcal{D}_u|} \sum_{i=1}^{|\mathcal{D}_u|} \left| \frac{\partial f_\theta(x, t)}{\partial t} + \mathcal{L}[f_\theta(x, t); \lambda] \right|_{\mathcal{D}_u}^2 + \frac{1}{|\mathcal{D}_0|} \sum_{i=1}^{|\mathcal{D}_0|} |f_\theta - u_0|_{\mathcal{D}_0}^2 + \frac{1}{|\mathcal{D}_b|} \sum_{i=1}^{|\mathcal{D}_b|} |f_\theta - g|_{\mathcal{D}_b}^2 ,$$

where $g(t, x)$ is the boundary condition and $u_0(x)$ is the initial condition.$\mathcal{D}_u, \mathcal{D}_b, \mathcal{D}_0$ denote the points set sampled from the bulk, the boundary condition and the initial condition respectively.

For the boundary-included model and initial-included model, The empirical loss $\hat{\mathcal{R}}$ can remove the corresponding term in the Vanilla empirical loss function:

$$\begin{aligned}\hat{\mathcal{R}}_{\text{boundary-included}} &= \frac{1}{|\mathcal{D}_u|}\sum_{i=1}^{|\mathcal{D}_u|}\left|\frac{\partial f_\theta(x,t)}{\partial t}+\mathcal{L}\left[f_\theta(x,t);\lambda\right]\right|_{\mathcal{D}_u}^2+\frac{1}{|\mathcal{D}_0|}\sum_{i=1}^{|\mathcal{D}_0|}\left|f_\theta-u_0\right|_{\mathcal{D}_0}^2\\ \hat{\mathcal{R}}_{\text{initial-included}} &= \frac{1}{|\mathcal{D}_u|}\sum_{i=1}^{|\mathcal{D}_u|}\left|\frac{\partial f_\theta(x,t)}{\partial t}+\mathcal{L}\left[f_\theta(x,t);\lambda\right]\right|_{\mathcal{D}_u}^2+\frac{1}{|\mathcal{D}_b|}\sum_{i=1}^{|\mathcal{D}_b|}\left|f_\theta-g\right|_{\mathcal{D}_b}^2\end{aligned}$$

3.3 Fractional Error

Fractional errors are often used in numerical methods for solving differential equations (Glusa & Otárola, 2021) to assess the accuracy of the numerical solution. And also it will provide a normalized measure for the difference between the predicted solution and the exact solution, so it can be used to compare the accuracy of different numerical methods or different parameter settings for a given method. In my experiment, I will use Fractional-Error to measure the numerical error, and the formulation will be below:

$$\text{Fractional-Error} = \frac{\|\hat{u}(\boldsymbol{x};\theta)-u(\boldsymbol{x})\|_2^2}{\|u(\boldsymbol{x})\|_2^2}$$

where $\hat{u}(\boldsymbol{x};\theta)$ is the predicted solution for a given input x and model parameters θ, $u(\boldsymbol{x})$ is the exact solution for the same input $\boldsymbol{x}$, and $\|\cdot\|_2$ represents the L2 norm. (Notes, 2014) Why do you generally use the L2 loss function, because it is easy to calculate.What's more, the value of each parameter when taking the minimum value can be obtained directly by derivation. In addition, there must be only one best prediction line with L2, and there may be multiple optimal solutions for L1 because of its nature.

Fractional Error is the ratio of the absolute difference between a predicted value and the actual value to the actual value, also it is the relative error. So in this way it is helpful for assessing the model's accuracy and finding the potential improvement areas.

3.4 The Stochastic Gradient Descent Algorithm

During the training process of neural networks, stochastic gradient descent (SGD)(Bottou, 2012) is a commonly used method for optimization. When the data size is relatively small, we can use these methods to calculate the gradient on all data and perform update iterations. And when the data size is relatively large, the overhead of calculating all data gradients each time will be huge. Because stochastic gradient descent can greatly reduce computational overhead, it is often used in large-scale data optimization. In this problem, I also choose the stochastic gradient descent method to minimize $I(u)$. This optimization process can be expressed as:

$$\theta^{k+1}=\theta^k-\eta\nabla_\theta\frac{1}{N}\sum_{i=1}^N g\left(x_{i,k},\theta^k\right),$$

where $\{x_{i,k}\}$ is chosen at random from an even grid of points. I employ a stochastic gradient descent technique with a mini-batch and an Adam optimizer(Kingma & Ba, 2014) which is widely used in deep learning tasks and thus is ubiquitous in the experiments reported in this reportcit

3.5 Double dencent

In traditional machine learning, in principle, we should choose a model of moderate size that matches the complexity of the problem.
For Figure 5, we can get that if the model is too small, it will be underfitting, and if the model is too large, it will be overfitting, which is not what we want. And in the practice of traditional machine learning, this principle is also widely recognized. This is what everyone learns when they get started with machine learning.
But this principle is wrong in deep learning. In deep learning, the relationship between Test Error and Model Complexity is a Double Descent curve that violates traditional machine learning understanding.

The Double Descent phenomenon was revealed by (Belkin et al., 2019) in 2018, and then it was more comprehensively studied in a recent ICLR2020 article (Nakkiran et al., 2021). The Double Descent curve in the experiment of article (Nakkiran et al., 2021) looks like this in the figure.

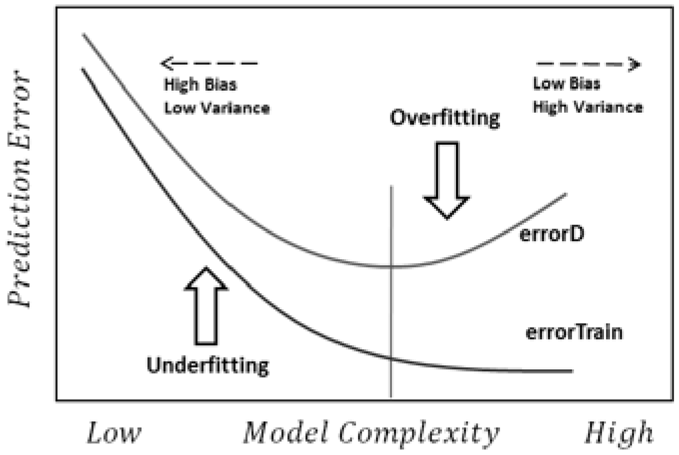


Figure 5: Model performance as complexity is varied

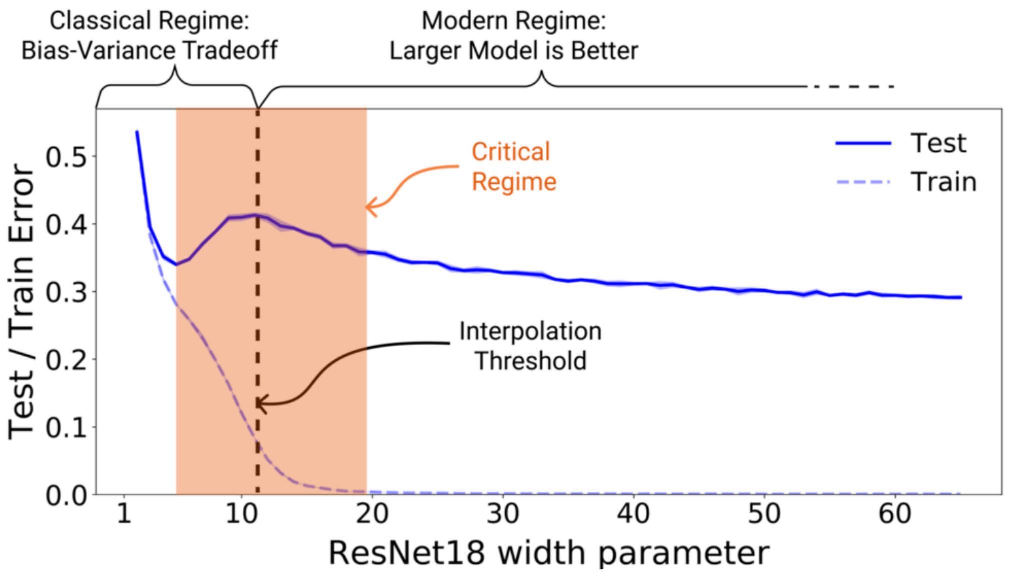


Figure 6: Double decent

We can clearly see that as the model parameters increase, Test Error first decreases, then increases, and then decreases for the second time.
The left side of the dotted line is the Bias-Variance Tradeoff is familiar to everyone in traditional machine learning, while the right side of the dotted line is the area where most deep learning models operate.
So, to put it simply, the answer to this question is - in principle, the larger the model, the better the accuracy at an acceptable cost.
Of course, in practice, the cost of model training and running also matters. Some scenarios also require the model to be small enough, fast enough, and low enough in energy consumption. In many places, accuracy is not paramount, and some trade-offs need to be made here.
Why does the model complexity increase, but the model generalization performance becomes better? The specific theoretical mechanism is still unclear. But this phenomenon is not limited to several specific model structures such as ResNet and Transformer. It is relatively certain that this phenomenon is closely related to the over-parameterization of the model and the stochastic optimization training method.
In the past few years, some statisticians have also discovered that the Double Descent phenomenon can also be reproduced (under some assumptions) in a simple high-dimensional statistical model Q, such as (Hastie

et al., 2022). This shows that Double Descent is not a black magic Q that belongs to the neural network Q, and there must be a very clever mathematical principle behind it to be explored.

4 Best model for Inviscid Burger Equation

Inviscid Burger Equation assumes zero viscosity and neglects the viscous force. (Matsumoto, 2021) tell us about the compressible Navier-Stokes equation, a nonlinear simultaneous PDE that comprises equations for the conservation of total energy, momentum, and mass. Since the original PINN is suggested as a rudimentary solution to the PDE, it needs to be examined if the PINN described here can be utilized to synchronize the PDE. As a result, as a preliminary investigation in this section, the analysis of the inviscid Burger equations (She et al., 1992) in one and two dimensions is conducted. The incompressible Navier-Stokes equation lacks a pressure gradient term, and the Burger equation is a nonlinear advection-diffusion equation.

4.1 One-dimensional Inviscid Burger Equation

The one-dimensional inviscid Burger equation, a partial differential equation, describes the development of fluid flow in one dimension without viscosity. So here, $d = 1$ and $\nu = 0$.

$$\partial_t \left[u_1\right] = \left[\partial_t u_1\right] \ , \ (\sum_{i=1}^{d} u_i \partial_{x_i}) \left[u_1\right] = \left[\textstyle\sum_{i=1}^{d} u_i \partial_{x_i} u_1\right] \ = \left[u_1 \partial_{x_1} u_1\right]$$

From the (Salih, 2015), we can get 1D Inviscid Burger equation is a special case of the nonlinear wave equation. The inial value problem, in this case, can be posed as:

$$\begin{gathered} \tfrac{\partial u}{\partial t} + u\tfrac{\partial u}{\partial x} = 0, \\ u(x,0) = F(x) \end{gathered}$$

Then, it is common in many applications of wave equations to consider initial conditions that are linear functions reason is that linear functions are relatively simple and easy to work with mathematically. So here, the initial condition can be written as $F(x) = \alpha x + \beta$. The equation above can be solved explicitly to yield the solution:

$$u(x,t) = \frac{\alpha x + \beta}{\alpha t + 1}$$

Then, we need to come back to the PDE and prove the solution is this PDE's solution:

$$\frac{\partial u}{\partial x} = \frac{\alpha}{\alpha t + 1}$$

$$\frac{\partial u}{\partial t} = \frac{-\alpha(\alpha x + \beta)}{(\alpha t + 1)^2}$$

$$\begin{aligned} &\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} \\ &= \frac{-\alpha(\alpha x + \beta)}{(\alpha t + 1)^2} + \frac{\alpha x + \beta}{\alpha t + 1} \cdot \frac{\alpha}{\alpha t + 1} \\ &= \frac{-\alpha(\alpha x + \beta)}{(\alpha t + 1)^2} + \frac{\alpha x + \beta}{\alpha(\alpha t + 1)} \\ &= 0 \end{aligned}$$

Now, we have proved that $u(x,t) = \frac{\alpha x + \beta}{\alpha t + 1}$ is one of the solutions for this Inviscid Burger Equation.

In order to state the example exact solution that we plan to check against and easy to calculate. I will choose $\alpha = 1$ and $\beta = 0$ in my experiments for 1D inviscid Burger Equation. The initial value problem in this case can be posed as $F(x) = x$, and the boundary condition can be posed as the exact solution. We can choose the x and t domains from 0 to 1. The formula for One-dimensional Inviscid Burger Equation is:

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = 0, (x,t) \in [0,1] \times [0,1]$$
$$u(x,0) = F(x) = x, x \in [0,1]$$
$$u(x,t) = g(x,t) = \frac{x}{t+1}, x \in \{0,1\}$$

4.1.1 Neural network for One-dimensional Inviscid Burger Equation

For One-dimensional Inviscid Burger Equation, in my experiment, it is just like $N_1 : \mathbb{R}^2 \to \mathbb{R}$, $(x,t) \to N_u$, where (x,t) is the input and N_u is the output. And the layer depth here is 5, so the details for this neural network are:

$\mathcal{N}(\boldsymbol{x}) := \mathrm{A}_4(\sigma \mathrm{A}_3(\sigma \mathrm{A}_2(\sigma \mathrm{A}_1(\boldsymbol{x}))))$, where

$\mathrm{A}_1 : \mathbb{R}^2 \to \mathbb{R}^w$, $\mathrm{A}_2 : \mathbb{R}^w \to \mathbb{R}^w$, $\mathrm{A}_3 : \mathbb{R}^w \to \mathbb{R}^w$, $\mathrm{A}_4 : \mathbb{R}^w \to \mathbb{R}$,

note: there are $2w^2 + 6w + 1$ parameters need to be trained in total in $\mathrm{N}(x)$

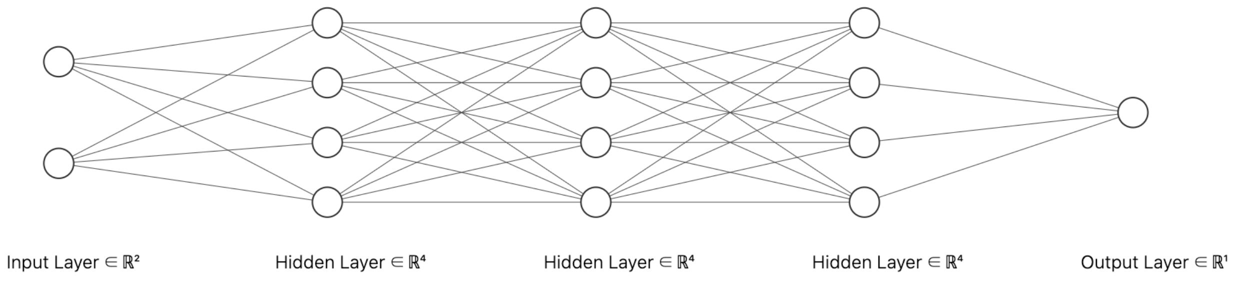


Figure 7: Neural network for One-dimensional Burger when width=4

4.1.2 Model for One-dimensional Inviscid Burger Equation

The vanilla model, which is trained solely on observational data without incorporating any additional physical constraints, can be useful when the underlying physics of the system is well-understood, and the focus is mainly on accurately fitting the data. However, in many cases, accurate prediction of the behavior of the system requires the incorporation of physical constraints.
Model for training on initial and boundary:

$$\mathrm{model}_{\mathrm{vanilla}}(x,t) := \mathrm{N}(x,t)$$

The boundary-included and initial-included models incorporate known boundary and initial conditions, respectively, into the training process. By doing so, these models are able to learn the underlying physics of the system, while also ensuring that the resulting predictions satisfy the known physical constraints. This approach can be particularly useful when the boundary or initial conditions are critical for determining the behavior of the system.

Model for training on boundary:

$$\mathrm{model}_{\mathrm{boundary-included}}(x,t) := \mathrm{N}(x,t) \cdot x \cdot (1-x) + (1-x) \cdot g(0,t) + x \cdot g(1,t)$$

The term $x \cdot (1-x)$ acts as a smooth function that enforces the boundary conditions, since it equals zero at both endpoints of the domain The other two terms in the model, $(1-x) \cdot g(0,t)$ and $x \cdot g(1,t)$, represent the known boundary conditions at the left and right endpoints of the domain, respectively. These terms are added to the neural network output to ensure that the resulting predictions satisfy the known physical constraints of the system.

Model for training on initial:

$$\mathrm{model}_{\mathrm{initial-included}}(x,t) := \mathrm{N}(x,t) \cdot t + F(x) \cdot (1-t)$$

The term t acts as a smooth function that enforces the initial condition, since it equals zero at the initial time point (where $t = 0$) and equals one at later time points (where $t > 0$).
The other term in the model, $F(x)\cdot(1-t)$, represents a "forcing term" or background solution that satisfies the PDE at all times. This term is added to the neural network output to ensure that the resulting predictions satisfy the known physical constraints of the system, while still allowing for arbitrary initial conditions.

4.1.3 Loss for One-dimensional Inviscid Burger Equation

These are the loss functions used for training the neural network models on initial, boundary, and initial/boundary data, respectively. In all three cases, the loss function includes a term that penalizes deviations from the PDE being solved, which is represented as the first item in each loss.

Loss for training on initial and boundary:

$$\hat{\mathcal{R}}_{\text{vanilla}}(\hat{u}) = \left\| \frac{\partial \hat{u}}{\partial t} + \hat{u}\frac{\partial \hat{u}}{\partial x} \right\|^2_{[0,1]\times[0,1],\nu_1} + \|\hat{u} - F(x)\|^2_{t=0,[0,1],\nu_2} + \|\hat{u} - g(x,t)\|^2_{\{0,1\}\times[0,1],\nu_3}$$

Loss for training on the boundary:

$$\hat{\mathcal{R}}_{\text{boundary-included}}(\hat{u}) = \left\| \frac{\partial \hat{u}}{\partial t} + \hat{u}\frac{\partial \hat{u}}{\partial x} \right\|^2_{[0,1]\times[0,1],\nu_1} + \|\hat{u} - F(x)\|^2_{t=0,[0,1],\nu_2}$$

Loss for training on the initial:

$$\hat{\mathcal{R}}_{\text{initial-included}}(\hat{u}) = \left\| \frac{\partial \hat{u}}{\partial t} + \hat{u}\frac{\partial \hat{u}}{\partial x} \right\|^2_{[0,1]\times[0,1],\nu_1} + \|\hat{u} - g(x,t)\|^2_{\{0,1\}\times[0,1],\nu_3}$$

where $\hat{u}$ is approximate solutions predicted as an output of DNN.

4.1.4 Different model for One-dimensional Inviscid Burger Equation

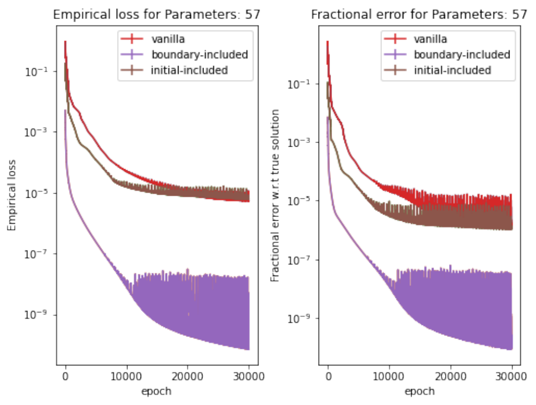
Different models for the Burger equation, including vanilla, boundary-included, and initial-included, are needed because they correspond to different physical scenarios and initial/boundary conditions that the equation can describe.
The vanilla model assumes that the system is homogeneous and has no external forces or boundary effects. This allows for a simple and generalized description of the behavior of solutions to the equation.
The boundary-included model involves incorporating the effect of boundaries on the system, which can greatly impact the dynamics of the solution. This model is useful for studying problems such as shock wave propagation in confined domains. The initial-included model takes into account the initial conditions of the system, which can also have a significant effect on the behavior of solutions. This model is particularly useful for analyzing problems where the initial state of the system is known but the boundary conditions are not well defined.
In summary, different models for the Burger equation allow us to study various physical scenarios and initial/boundary conditions, enabling a more comprehensive understanding of the system's behavior.
For conducting various experiments for different models with different parameters in the neural network is important to understand and compare the effects of these parameters on the performance of the model for solving the Burger equation.
By systematically varying these parameters and comparing the results across different models, researchers can gain insights into the strengths and weaknesses of different approaches to solving the Burger equation. This can help guide the development of more efficient and accurate numerical methods for solving this important nonlinear partial differential equation.
Since I'm doing five replicates here, I'll take the mean and then figure out the variance (the variance of the model may increase as the number of parameters increases, making the model more sensitive to noise in the input data. This can result in the model becoming overly complex and harder to interpret).

In Table 1, each experiment is labeled Experiment1 to Experiment4 and corresponds to a different value of the parameter p. The first column of the table lists the experiment number and a corresponding figure label.

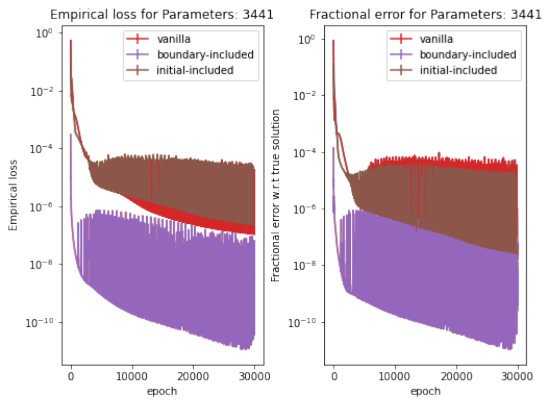
The second column lists the number of parameters used in each experiment. The third column provides details on the mesh size used in each experiment.

1D Burger, epoch =30000, lr=0.0001, p is parameter number		
figure 8	parameter	mesh size
Experiment1:[$p < n$]	57	**vanilla**:bulk mesh size=1000,initial mesh size=1000, boundary mesh size=1000 **boundary-included**: bulk mesh size = 1000,initial mesh size = 1000, boundary mesh size = 0, **initial-included**: bulk mesh size = 1000,initial mesh size = 0, boundary mesh size = 1000(500 for x=1 and 500 for x=0)
Experiment2:[$p \approx n$]	3441	
Experiment3:[$p \geq n$]	81201	
Experiment4:[$p \gg n$]	181801	

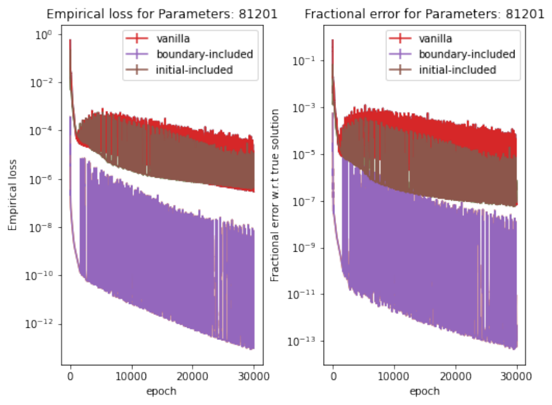
Table 1: 1D Inviscid Burger equation experiments with different parameter(n is the mesh size for Vanilla)



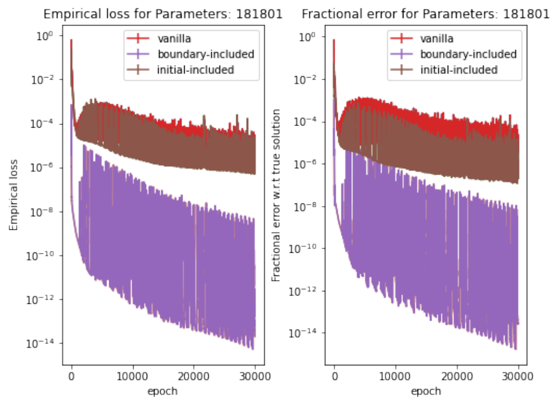
(a) Results for the 57 parameters net



(b) Results for the 3441 parameters net



(c) Results for the 81201 parameters net



(d) Results for the 181801 parameters net

Figure 8: 1D Inviscid Burger, epoch = 30000, lr = 0.0001,mesh size=3000

From the above figure 8, we can conclude that *no matter how large our number of parameters is, the boundary-included model will always have the best performance. When the number of parameters gradually increases, it is not surprising to find that both loss and error have decreased.*

In fact, we can also find that compared with the vanilla model, initial-included also has a small optimization, but the effect is not very obvious.

4.1.5 Plot the heatmap for 1D Inviscid Burger Equation

Heatmap is a very useful way to visualize data presented in a grid or matrix format. In the case of solving partial differential equations (PDEs) using deep learning, a heatmap can be used to visualize the predicted and true solutions of the PDE.

By plotting the predicted and true solutions side-by-side in a heatmap, we can easily compare the two and identify any areas where the predicted and true solutions differ significantly. This can help us understand how our neural network model is performing and can guide us to improve the model if necessary.
Heatmap is particularly useful for visualizing PDE solutions with multiple dimensions, such as solutions over time and space. By plotting these solutions on a heatmap, we can get a clear picture of how the solutions change over time and space and can identify any patterns or trends that might be important for understanding the behavior of the system being modeled.

For figure 9 and figure 10, both these two figures top left is the exact solution, the top right is the Vanilla model prediction, the bottom left is Boundary-included model prediction, and the bottom right is Initial-included model prediction.
In fact, the performance of these three models is all fantastic, because they are very similar to the exact solution. However, we can also clearly see that Vanilla's performance is not as good as the other two models by observing the bar beside the heatmap.

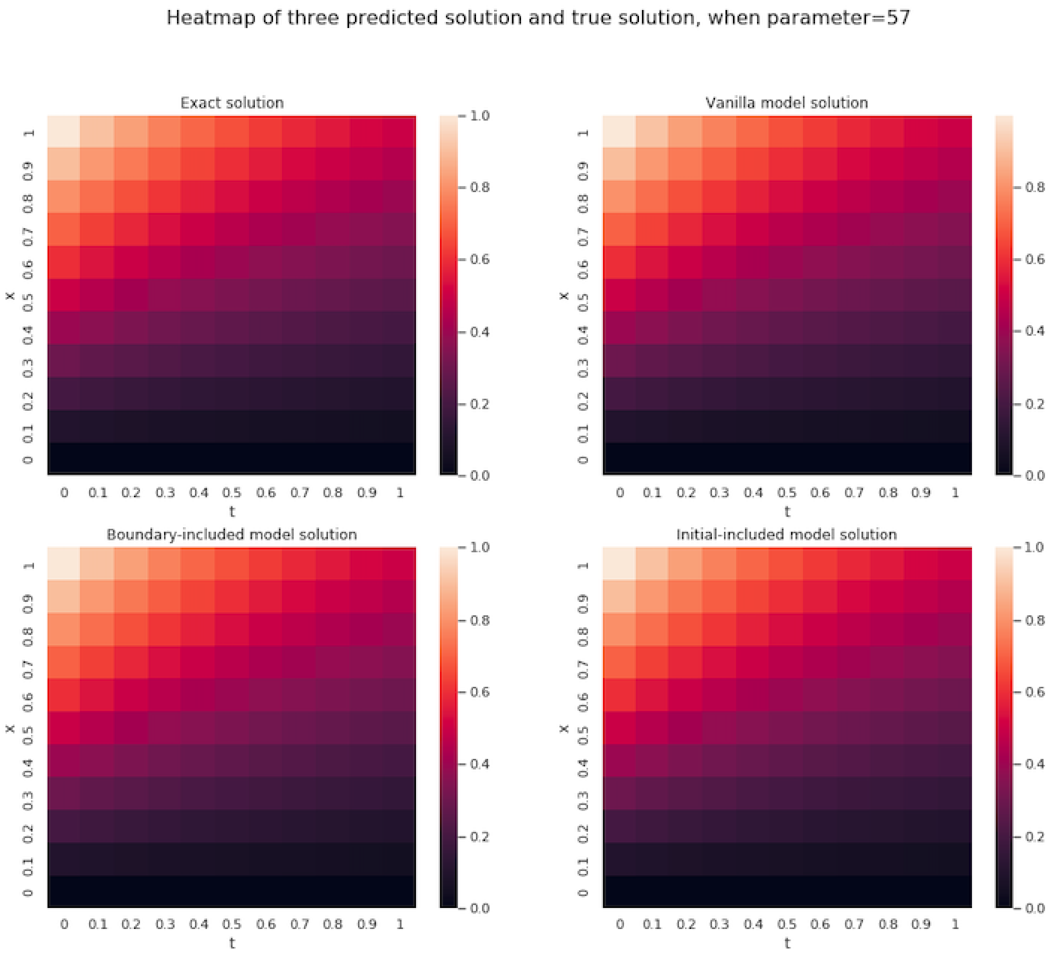


Figure 9: Heatmeap for 1D Inviscid Burger Equation when parameter=57

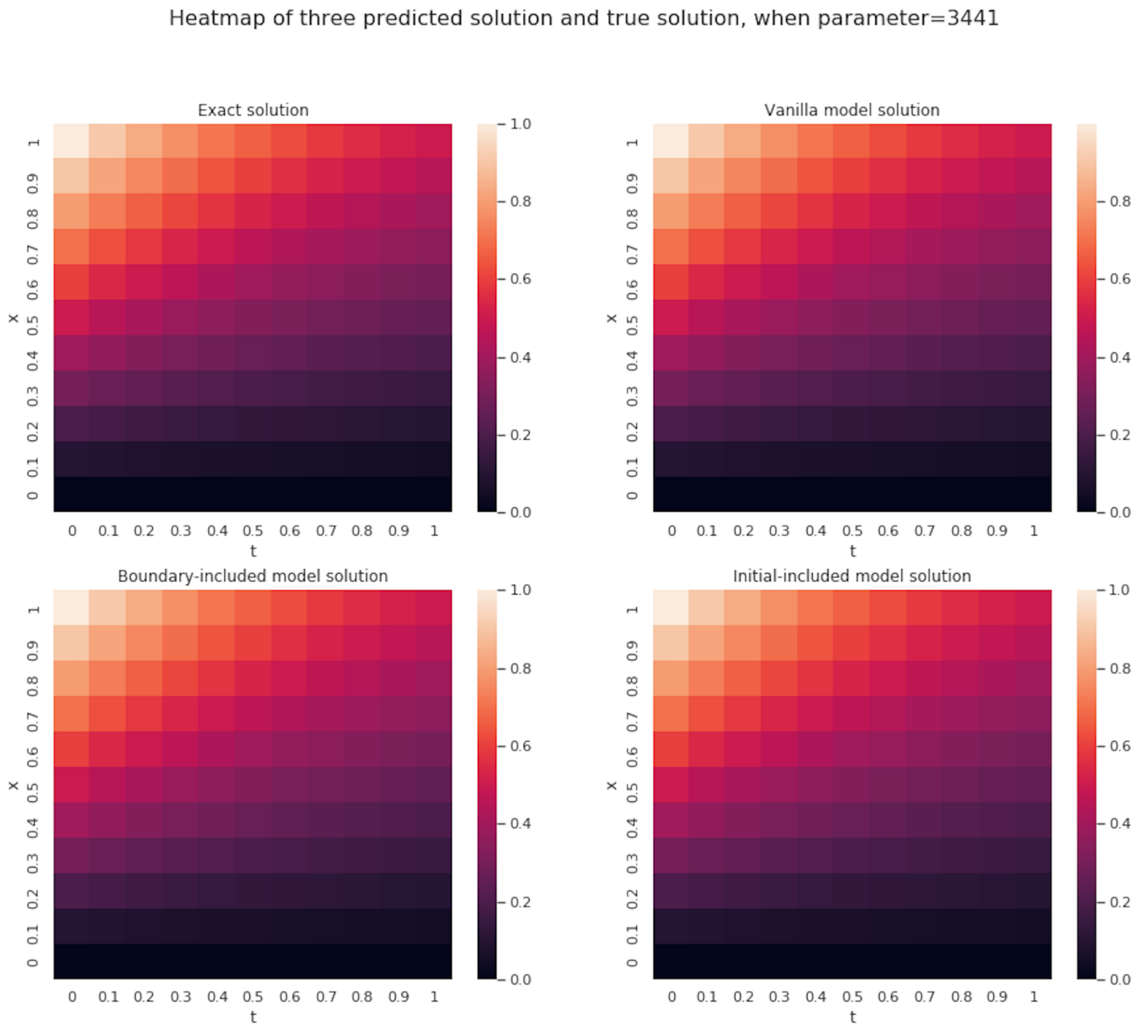


Figure 10: Heatmeap for 1D Inviscid Burger Equation when parameter=3441

4.1.6 Time and device

Time here is the total time for 3 different models, and for each set of experiments, I repeated five times.

1D Inviscid Burger, epoch =30000, lr=0.0001, mesh size=3000		
Parameter	Device	Time
Parameter=57	NVIDIA GeForce RTX 3080	33.5 mins
Parameter=3441	NVIDIA GeForce RTX 3080	60.8 mins
Parameter=81201	NVIDIA GeForce RTX 3080	528.7 mins
Parameter=181801	NVIDIA GeForce RTX 3080	1206.3 mins

Table 2: Time and device for 1D Inviscid Burger equation experiments with different parameter

4.2 Two-dimensional Inviscid Burger Equation

Two-dimensional Inviscid Burger equations are formed by multidimensional Burger equations. For here, $d = 2$ and $\nu = 0$.

$$\partial_t \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \partial_t u_1 \\ \partial_t u_2 \end{bmatrix} \;,\; (\sum_{i=1}^{d} u_i \partial_{x_i}) \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{d} u_i \partial_{x_i} u_1 \\ \sum_{i=1}^{d} u_i \partial_{x_i} u_2 \end{bmatrix} = \begin{bmatrix} u_1 \partial_{x_1} u_1 + u_2 \partial_{x_2} u_1 \\ u_1 \partial_{x_1} u_2 + u_2 \partial_{x_2} u_2 \end{bmatrix}$$

The governing equations targeted and the Two-dimensional Inviscid Burger equation are as follows.

$$\begin{cases} u_t + u u_x + v u_y = 0 \\ v_t + u v_x + v v_y = 0 \end{cases}$$

I find the paper(Zhu et al., 2010) which can give me one of the exact solutions for the 2D Inviscid Burger equation. The initial value problem, in this case, can be posed as:

$$\begin{cases} u_0 = x + y \\ v_0 = x - y \end{cases}$$

In the following example, we consider the 2D Burger equations, with the initial conditions above, and the exact solutions are as follows:

$$\begin{cases} u(x,y,t) = \frac{x+y-2\cdot x\cdot t}{1-2\cdot t^2} \\ v(x,y,t) = \frac{x-y-2\cdot y\cdot t}{1-2\cdot t^2} \end{cases}$$

I chose to use the exact solution as a reference to evaluate the accuracy of my model's predictions at the boundaries. By applying the exact solution at the boundary points, I was able to obtain the initial and boundary conditions necessary to train and test my model.
At this time I choose the x, and y domains from 0 to 1 and the t domains from 0 to 0.5. The initial conditions and boundary conditions applied in this Two-dimensional Inviscid Burger Equation are:

$$\begin{cases} u_t + u u_x + v u_y = 0 \\ v_t + u v_x + v v_y = 0 \end{cases} , \begin{cases} u_0 = x + y \\ v_0 = x - y \end{cases} , \begin{cases} g_{x,0}(y,t) = \frac{y}{1-2\cdot t^2} \\ g_{x,1}(y,t) = \frac{1-y-2\cdot t}{1-2\cdot t^2} \\ g_{y,0}(x,t) = \frac{x}{1-2\cdot t^2} \\ g_{y,1}(x,t) = \frac{1-x-2\cdot t}{1-2\cdot t^2} \end{cases}$$

where $g_{x,0}(y,t)$ and $g_{x,1}(y,t)$ is the boundary condition for u when $x = 0$ and $x = 1$, $g_{y,0}(x,t)$ and $g_{y,1}(x,t)$ is the boundary condition for v when $y = 0$ and $y = 1$, u_0 and v_0 is the initial condition. In the next section, I will try to prove the solution.

4.2.1 Two-dimensional Inviscid Burger Equation's solution

Multi-dimensional Burger equations form into PDEs. The governing equations targeted here are as follows.

$$\begin{cases} u_t + u u_x + v u_y = 0 \\ v_t + u v_x + v v_y = 0 \end{cases}$$

Define the exact solution is :

$$\left.\begin{aligned} u(x,y,t) &= \tfrac{x+y-2\cdot x\cdot t}{1-2\cdot t^2}, \\ v(x,y,t) &= \tfrac{x-y-2\cdot y\cdot t}{1-2\cdot t^2}, \end{aligned}\right\} (\mathrm{x},\mathrm{y}) \in \Omega,$$

The computational domain is assumed to be square. $\Omega = \{(x,y) : 0 \le x \le 1, 0 \le y \le 1\}$. In this problem, the initial and boundary conditions for $u(x,y,t)$ and $v(x,y,t)$ have been taken from the above analytical solutions.

Then, we need to come back to the PDE and prove the solution is this PDE's solution:

$$u_t = \frac{x(-4t^2 + 4t - 2) + 4y\cdot t}{(1-2t^2)^2}$$

$$v_t = \frac{-2(-2x\cdot t - 2y\cdot t^2 + 2y\cdot t + y)}{(1-2t^2)^2}$$

$$u_x = \frac{1-2t}{1-2t^2}, u_y = \frac{1}{1-2t^2}$$
$$v_x = \frac{1}{1-2t^2}, v_y = \frac{-1-2t}{1-2t^2}$$
$$u_{xx} = 0, u_{yy} = 0, v_{xx} = 0, v_{yy} = 0$$

and we need to prove:

$$\begin{cases} u_t + uu_x + vu_y = 0 \\ v_t + uv_x + vv_y = 0 \end{cases}$$

$$\begin{aligned}
&u_t + uu_x + vu_y \\
&= \frac{x(-4t^2+4t-2)+4y\cdot t}{(1-2t^2)^2} + \frac{x+y-2\cdot x\cdot t}{1-2\cdot t^2}\cdot\frac{1-2t}{1-2t^2} + \frac{x-y-2\cdot y\cdot t}{1-2\cdot t^2}\cdot\frac{1}{1-2t^2} \\
&= \frac{-4x\cdot t^2+4x\cdot t-2x+4y\cdot t+x+y-2x\cdot t-2y\cdot t-2x\cdot t+4x\cdot t^2+x-y-2y\cdot t}{(1-2t^2)^2} \\
&= \frac{-4x\cdot t^2+4x\cdot t^2+4x\cdot t-2x\cdot t-2x\cdot t-2x+x+x+4y\cdot t-2y\cdot t-2y\cdot t+y-y}{(1-2t^2)^2} \\
&= 0
\end{aligned}$$

$$\begin{aligned}
&v_t + uv_x + vv_y \\
&= \frac{-2(-2x\cdot t-2y\cdot t^2+2y\cdot t+y)}{(1-2t^2)^2} + \frac{x+y-2\cdot x\cdot t}{1}\cdot\frac{1-2t}{1-2t^2} + \frac{x-y-2\cdot y\cdot t}{1-2\cdot t^2}\cdot\frac{-1-2t}{1-2t^2} \\
&= \frac{4x\cdot t-4y\cdot t^2-4y\cdot t-2y+x+y-2x\cdot t-x+y+2y\cdot t-2x\cdot t+2y\cdot t+4y\cdot t^2}{(1-2t^2)^2} \\
&= \frac{4x\cdot t-2x\cdot t-2x\cdot t-4y\cdot t^2+4y\cdot t^2-4y\cdot t+2y\cdot t+2y\cdot t-2y+y+y+x-x}{(1-2t^2)^2} \\
&= 0
\end{aligned}$$

Now, we have proved that $u(x,y,t) = \frac{x+y-2\cdot x\cdot t}{1-2\cdot t^2}, v(x,y,t) = \frac{x-y-2\cdot y\cdot t}{1-2\cdot t^2}$ is the solution for Two-dimensional Inviscid Burger equation.

4.2.2 Neural network for Two-dimensional Burger Equation

For Two-dimensional Inviscid Burger Equation, in my experiment, it is just like $N_1 : \mathbb{R}^3 \to \mathbb{R}^2$, $(x,y,t) \to N_{u,v}$, where (x,y,t) is the input and $N_{u,v}$ is the output. And the layer depth here is 5, so the details for this neural network are:
$\mathcal{N}(\boldsymbol{x}) \coloneqq \mathrm{A}_4(\sigma\mathrm{A}_3(\sigma\mathrm{A}_2(\sigma\mathrm{A}_1(\boldsymbol{x}))))$, where

$\mathrm{A}_1 : \mathbb{R}^3 \to \mathbb{R}^w$, $\mathrm{A}_2 : \mathbb{R}^w \to \mathbb{R}^w$, $\mathrm{A}_3 : \mathbb{R}^w \to \mathbb{R}^w$, $\mathrm{A}_4 : \mathbb{R}^w \to \mathbb{R}^2$, note: there are $2w^2 + 8w + 2$ parameters need to be trained in total in $\mathrm{N}(x)$

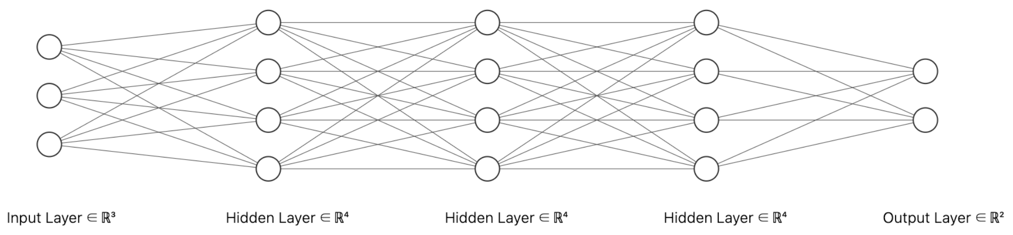


Figure 11: Neural network for Two-dimensional Burger equation when width=4

4.2.3 Model for Two-dimensional Inviscid Burger Equation

Model for training on initial and boundary:

$$\text{model}_{\text{vanilla-u}}(x,y,t) := \mathrm{N_u}(x,y,t)$$

$$\text{model}_{\text{vanilla-v}}(x,y,t) := \mathrm{N_v}(x,y,t)$$

Model for training on the boundary:

$$\text{model}_{\text{boundary-included-u}}(x,y,t) := \mathrm{N_u}(x,y,t)\cdot x\cdot(1-x) + (1-x)\cdot g_{x,0}(y,t) + x\cdot g_{x,1}(y,t)$$

$$\text{model}_{\text{boundary-included-v}}(x,y,t) := \mathrm{N_v}(x,y,t)\cdot y\cdot(1-y) + (1-y)\cdot g_{y,0}(x,t) + y\cdot g_{y,1}(x,t)$$

Model for training on the initial:

$$\text{model}_{\text{initial-included-u}}(x,y,t) := \mathrm{N_u}(x,y,t)\cdot t + 2\cdot(0.5-t)\cdot u_0(x,y)$$

$$\text{model}_{\text{initial-included-v}}(x,y,t) := \mathrm{N_v}(x,y,t)\cdot t + 2\cdot(0.5-t)\cdot v_0(x,y)$$

4.2.4 Loss for Two-dimensional Inviscid Burger Equation

$\hat{\mathcal{R}}1$ and $\hat{\mathcal{R}}_2$ that are parts of the loss function correspond to the residual of the PDE.

$$\begin{cases} \hat{\mathcal{R}}_1 = \left\| \dfrac{\partial \hat{u}}{\partial t} + \hat{u}\dfrac{\partial \hat{u}}{\partial x} + \hat{v}\dfrac{\partial \hat{u}}{\partial y} \right\|^2_{[0,1]^2\times[0,1],\nu_1} \\ \hat{\mathcal{R}}_2 = \left\| \dfrac{\partial \hat{v}}{\partial t} + \hat{u}\dfrac{\partial \hat{v}}{\partial x} + \hat{v}\dfrac{\partial \hat{v}}{\partial y} \right\|^2_{[0,1]^2\times[0,1],\nu_1} \end{cases}$$

$\hat{\mathcal{R}}_3$ and $\hat{\mathcal{R}}_4$ correspond to the evaluation of boundary conditions.

$$\begin{cases} \hat{\mathcal{R}}_3 = \|\hat{u} - g,0_x(y,t)\|^2_{\{0\}\times[0,1]\times[0,1],\nu_2} + \|\hat{u} - g_{x,1}(y,t)\|^2_{\{1\}\times[0,1]\times[0,1],\nu_2} \\ \hat{\mathcal{R}}_4 = \|\hat{v} - g_{y,0}(x,t)\|^2_{[0,1]\times\{0\}\times[0,1],\nu_2} + \|\hat{v} - g_{y,1}(x,t)\|^2_{[0,1]\times\{1\}\times[0,1],\nu_2} \end{cases}$$

$\hat{\mathcal{R}}_5$ and $\hat{\mathcal{R}}_6$ correspond to the evaluation of initial conditions,

$$\begin{cases} \hat{\mathcal{R}}_5 = \|\hat{u} - u_0\|^2_{[0,1]^2,t=0,\nu_3} \\ \hat{\mathcal{R}}_6 = \|\hat{v} - v_0\|^2_{[0,1]^2,t=0,\nu_3} \end{cases}$$

where $\hat{u}$ and $\hat{v}$ are approximate solutions predicted as an output of DNN.

Loss for training on initial and boundary:

$$\hat{\mathcal{R}}_{\text{vanilla}} = \left(\hat{\mathcal{R}}_1 + \hat{\mathcal{R}}_2\right) + \left(\hat{\mathcal{R}}_3 + \hat{\mathcal{R}}_4\right) + \left(\hat{\mathcal{R}}_5 + \hat{\mathcal{R}}_6\right)$$

Loss for training on the boundary:

$$\hat{\mathcal{R}}_{\text{boundary-included}} = \left(\hat{\mathcal{R}}_1 + \hat{\mathcal{R}}_2\right) + \left(\hat{\mathcal{R}}_5 + \hat{\mathcal{R}}_6\right)$$

Loss for training on the initial:

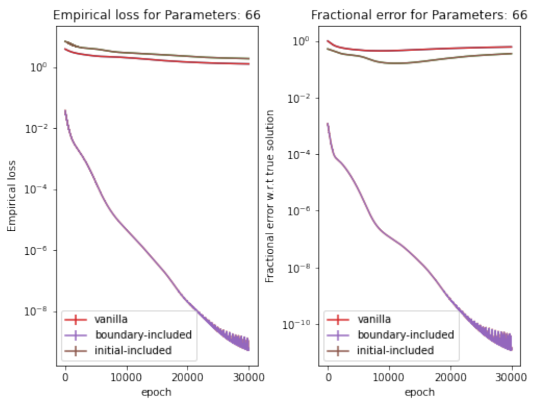
$$\hat{\mathcal{R}}_{\text{initial-included}} = \left(\hat{\mathcal{R}}_1 + \hat{\mathcal{R}}_2\right) + \left(\hat{\mathcal{R}}_3 + \hat{\mathcal{R}}_4\right)$$

2D Burger, epoch =30000, lr=0.001, p is parameter number		
figure 12	parameter	mesh size
Experiment1:$[p < n]$	66	**vanilla**:bulk mesh size=800,initial mesh size=800, boundary mesh size=800; **boundary-included**: bulk mesh size=800,initial mesh size=800, boundary mesh size=0; **initial-included**: bulk mesh size = 800, initial mesh size = 0, boundary mesh size = 800
Experiment2:$[p \approx n]$	1794	
Experiment3:$[p \geq n]$	20802	

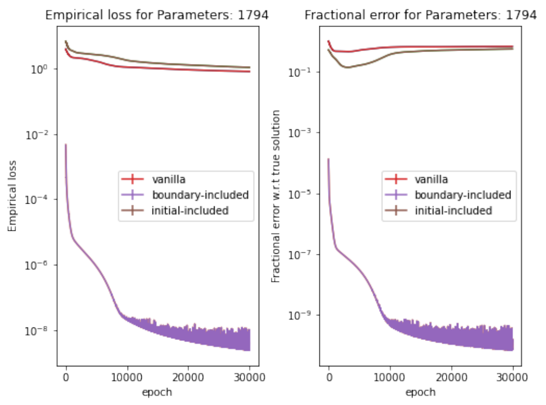
Table 3: 2D Inviscid Burger equation experiments with different parameter(n is the mesh size for Vanilla)

4.2.5 Different model for Two-dimensional Inviscid Burger Equation

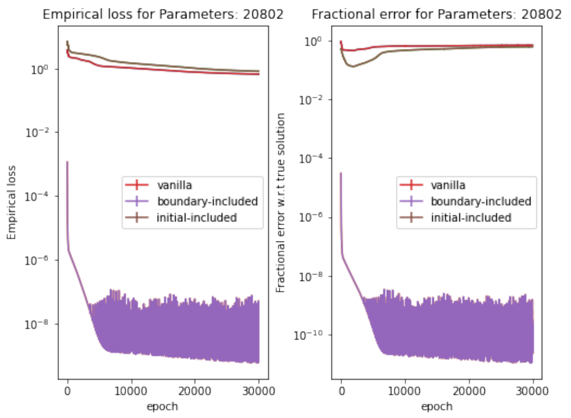
In Table 3, a distinct value of the parameter p corresponds to each experiment, which is designated Experiments 1 through 4. The experiment number and a related figure label are included in the first column of the table. The number of parameters utilized in each experiment is listed in the second column. Information on the mesh size used in each experiment is provided in the third column. To be noticed, for the boundary mesh size, it will contain 200 points for $x = 0$, 200 points for $x = 1$, 200 points for $y = 0$, 200 points for $y = 1$



(a) Results for the 66 parameters net



(b) Results for the 1794 parameters net



(c) Results for the 20802 parameters net

Figure 12: 2D Burger, epoch = 30000, lr = 0.001,mesh size=2400

Similarly, we found that the conclusion here is similar to that of 1D Inviscid Burger, and *the boundary-included model has the best effect, which greatly improves the accuracy rate of 2D Inviscid Burger Equation.*

4.2.6 Time and device

Time here is the total time for 3 different models, and for each set of experiments, I repeated three times.

2D Inviscid Burger, epoch =30000, lr=0.001, mesh size=3000		
Parameter	Device	Time
Parameter=66	NVIDIA Tesla V100	96.1 mins
Parameter=1794	NVIDIA Tesla V100	103.6 mins
Parameter=20802	NVIDIA Tesla V100	112.9 mins

Table 4: Time and device for 2D Inviscid Burger equation experiments with different parameter

4.3 Conclusion for Inviscid Burger Equation

For the Inviscid Burger equation, I did one-dimensional and two-dimensional experiments to test three different types of models. After a long period of training, I conclude that no matter what dimension or neural network size, the boundary-included model will give the best exact solution. Especially in the two-dimensional case, neither the vanilla model nor the initial-included model gives very accurate solutions and the boundary-included model greatly improves the performance.

5 Best model for Viscous Burger Equation

In the previous discussion, I only talk about the Burger equation when ν equals zero and it is the Inviscid Burger equation. However, the Burger equation is widely used in various fields, including fluid mechanics, statistical physics, and nonlinear dynamics. And $\vec{x}$ is the velocity vector of the fluid, t is time, and ν is the kinematic viscosity coefficient of the fluid.
The term on the left-hand side of the equation represents the advection of the fluid by its own velocity, while the term on the right-hand side represents the diffusion of the fluid due to its viscosity.

The presence of viscosity (as represented by the ν term) is crucial for the stability of the solutions of the Burger equation. Without viscosity, the solutions would develop singularities very quickly and become physically meaningless. In summary, the viscosity term in the Burger equation ensures that the solutions of the equation are physically meaningful and stable.

So I decide to choose $\nu = 0.01$, $\nu = 0.1$, $\nu = 0.5$, $\nu = 0.1$, $\nu = 2$ and $\nu = 10$ in the experiment for 1D Viscous Burger equation. And due to the time limitation I will choose $\nu = 0.01$, $\nu = 0.1$, $\nu = 0.5$, $\nu = 0.1$, $\nu = 2$ and $\nu = 10$ in the experiment for 3D Viscous Burger equation. For model is the same as before: Vanilla, boundary-included, and initial-included. And for the neural network layer width is 4, 40, and 300 for 1D, 4, and 40 for 3D.

5.1 One-dimensional Viscous Burger Equation

The one-dimensional viscous Burger equation is a partial differential equation that describes the evolution of a velocity field in a fluid with viscosity. So here, $d = 1$ and $\nu > 0$.

$$\partial_t \left[u_1\right] = \left[\partial_t u_1\right] \ , \ (\sum_{i=1}^{d} u_i \partial_{x_i}) \left[u_1\right] = \left[\textstyle\sum_{i=1}^{d} u_i \partial_{x_i} u_1\right] \ = \left[u_1 \partial_{x_1} u_1\right], \ \nu \left[\nabla^2 u_1\right] = \nu \left[\textstyle\sum_{i=1}^{d} \partial_{x_i}^2 u_1\right] = \nu \left[\partial_{x_1}^2 u_1\right]$$

So the formulation for 1D viscid Burger equation can be posed as ($F(x)$ is the initial condition and $g(x,t)$ is the boundary condition):

$$\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}$$
$$u(x,0) = F(x)$$
$$u(x,t) = g(x,t)$$

From the (Agarwal, 2011), we can get one of the exact solution for 1D viscid Burger equation:

$$u(x,t) = 1 - \tanh \frac{x - x_c - t}{2\nu}$$

However in the paper, I can not get the boundary condition and initial condition, and we know that using the exact solution in this way is often done in deep learning-based approaches to solving partial differential equations. We can choose the x and t domains from 0 to 1. The formula for One-dimensional viscid Burger Equation for some $x_c \in \mathbb{R}$ is:

$$\begin{gathered}\frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} = \nu \frac{\partial^2 u}{\partial x^2}, (x,t) \in [0,1] \times [0,1] \\ u(x,0) = F(x) = 1 - \tanh \frac{x-x_c}{2\nu}, x \in [0,1] \\ u(x,t) = g(x,t) = 1 - \tanh \frac{x-x_c-t}{2\nu}, x \in \{0,1\}\end{gathered}$$

For the neural network, the model and loss function will be the same as the 1D Inviscid Burger equation.

5.1.1 Prove the solution

To check if the solution above is the true solution, recall:

$$\begin{aligned}\frac{d}{dx}(\tanh x) &= \operatorname{sech}^2 x \\ \frac{d^2}{dx^2}(\tanh x) &= -2\operatorname{sech}^3 x \sinh x\end{aligned}$$

Thus, using the above, we get:

$$\begin{aligned}\frac{\partial u(x,t)}{\partial t} &= -\frac{1}{2\nu}\operatorname{sech}^2 \frac{x-x_c-t}{2\nu} \\ \frac{\partial u(x,t)}{\partial x} &= \frac{1}{2\nu}\operatorname{sech}^2 \frac{x-x_c-t}{2\nu}\end{aligned}$$

Now,

$$\begin{aligned}\frac{\partial u(x,t)}{\partial t} + u(x,t)\frac{\partial u(x,t)}{\partial x} &= \frac{1}{2\nu}\operatorname{sech}^2\left(\frac{x-x_c-t}{2\nu}\right)(-1+u(x,t)) \\ &= -\frac{1}{2\nu}\operatorname{sech}^3\left(\frac{x-x_c-t}{2\nu}\right)\sinh\left(\frac{x-x_c-t}{2\nu}\right).\end{aligned}$$

Next,

$$\begin{aligned}\nu\frac{\partial^2 u(x,t)}{\partial x^2} &= \nu\left[\frac{1}{4\nu^2}(-2)\operatorname{sech}^3\left(\frac{x-x_c-t}{2\nu}\right)\sinh\left(\frac{x-x_c-t}{2\nu}\right)\right] \\ &= -\frac{1}{2\nu}\operatorname{sech}^3\left(\frac{x-x_c-t}{2\nu}\right)\sinh\left(\frac{x-x_c-t}{2\nu}\right)\end{aligned}$$

Thus, we can prove that it is a true solution.

5.1.2 Same ν, different model

The three tables (table 5,table 6 and table 7) below give the experimental results for solving a 1D viscous Burger equation with different values of viscosity parameter ν and different models (vanilla, boundary-included, and initial-included).

The experiments will contain some hyper-parameters: a fixed number of epochs (30000), learning rate (0.0001), mesh size (3000), and parameter (57, 3441, or 181,801 depending on the table). These hyper-parameters will affect the accuracy and speed of the model training and prediction and may need to be adjusted for different problems. For the variable, the viscosity parameter ν is varied in the experiments to test the performance of the model under different physical conditions. Different values of ν can affect the smoothness and complexity of the solution and may require different models or hyper-parameters to achieve good accuracy.

From the experiments previous, we can get that the choice of model configuration has a significant impact on the performance of the deep learning model in predicting the solution to the Burger equation. So in these experiments, I want to discuss two problems: 1. The accuracy of the prediction PDE with different behavior of fluid flow with viscosity parameter (ν); 2. The accuracy of the prediction PDE with different model configurations.

1D Burger, epoch =30000, lr=0.0001, mesh size=3000,parameter=57		
figure 13	ν	model
Experiment1	ν=0.01	**vanilla**:bulk mesh size=1000,initial mesh size=1000, boundary mesh size=1000 **boundary-included**: bulk mesh size = 1000,initial mesh size = 1000, boundary mesh size = 0, **initial-included**: bulk mesh size = 1000,initial mesh size = 0, boundary mesh size = 1000
Experiment2	ν=0.1	
Experiment3	ν=0.5	
Experiment4	ν=1	
Experiment5	ν=2	
Experiment6	ν=10	

Table 5: 1D Burger equation experiments with different values of ν when parameter=57

1D Burger, epoch =30000, lr=0.0001, mesh size=3000,parameter=3441		
figure 14	ν	model
Experiment1	ν=0.01	**vanilla**:bulk mesh size=1000,initial mesh size=1000, boundary mesh size=1000 **boundary-included**: bulk mesh size = 1000,initial mesh size = 1000, boundary mesh size = 0, **initial-included**: bulk mesh size = 1000,initial mesh size = 0, boundary mesh size = 1000
Experiment2	ν=0.1	
Experiment3	ν=0.5	
Experiment4	ν=1	
Experiment5	ν=2	
Experiment6	ν=10	

Table 6: 1D Burger equation experiments with different values of ν when parameter=3441

1D Burger, epoch =30000, lr=0.0001, mesh size=3000,parameter=181801		
figure 15	ν	model
Experiment1	ν=0.01	**vanilla**:bulk mesh size=1000,initial mesh size=1000, boundary mesh size=1000 **boundary-included**: bulk mesh size = 1000,initial mesh size = 1000, boundary mesh size = 0, **initial-included**: bulk mesh size = 1000,initial mesh size = 0, boundary mesh size = 1000
Experiment2	ν=0.1	
Experiment3	ν=0.5	
Experiment4	ν=1	
Experiment5	ν=2	
Experiment6	ν=10	

Table 7: 1D Burger equation experiments with different values of ν when parameter=181801

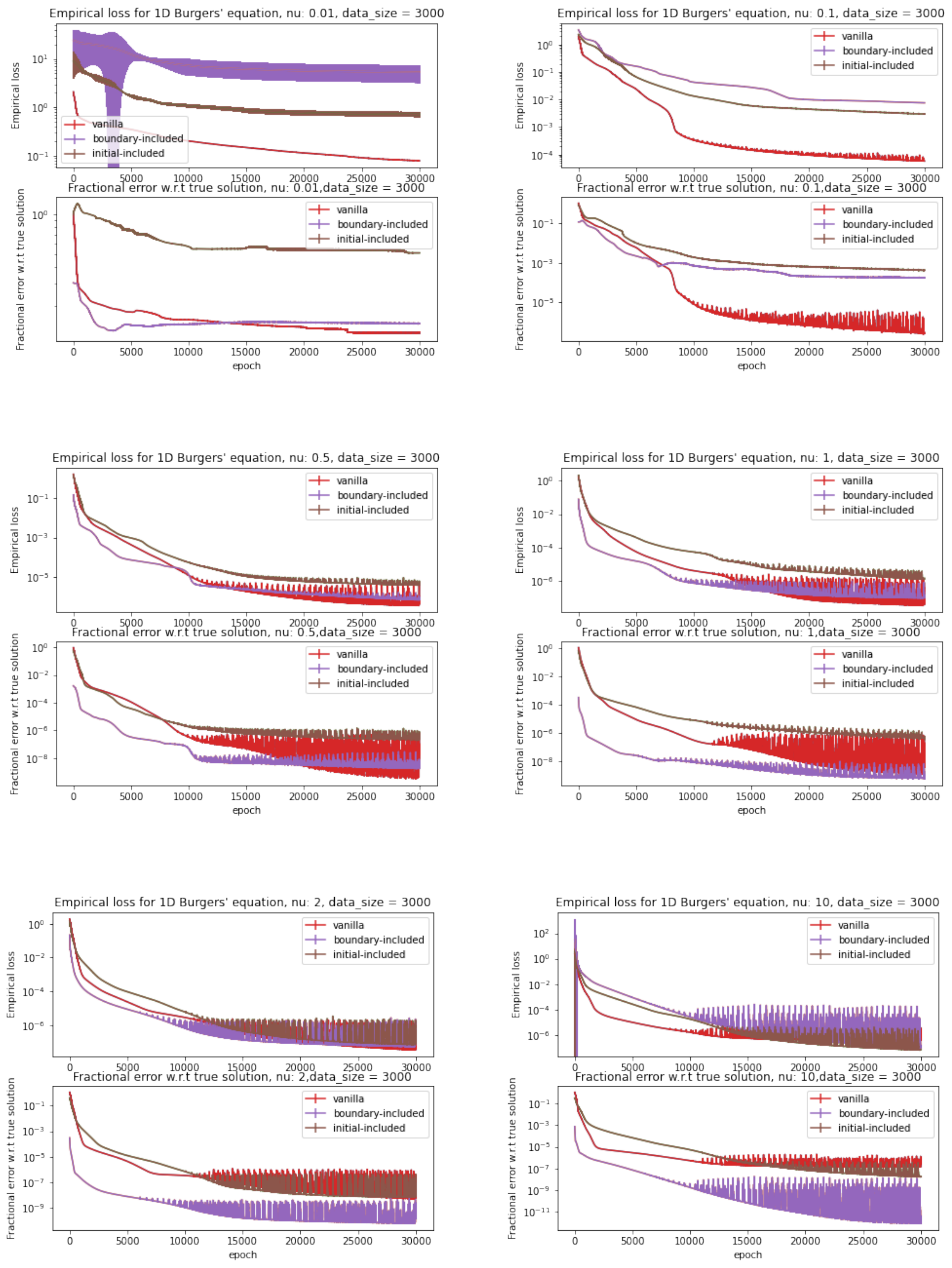


Figure 13: 1D Burger, epoch = 30000, para: 57, lr = 0.0001,mesh size=3000

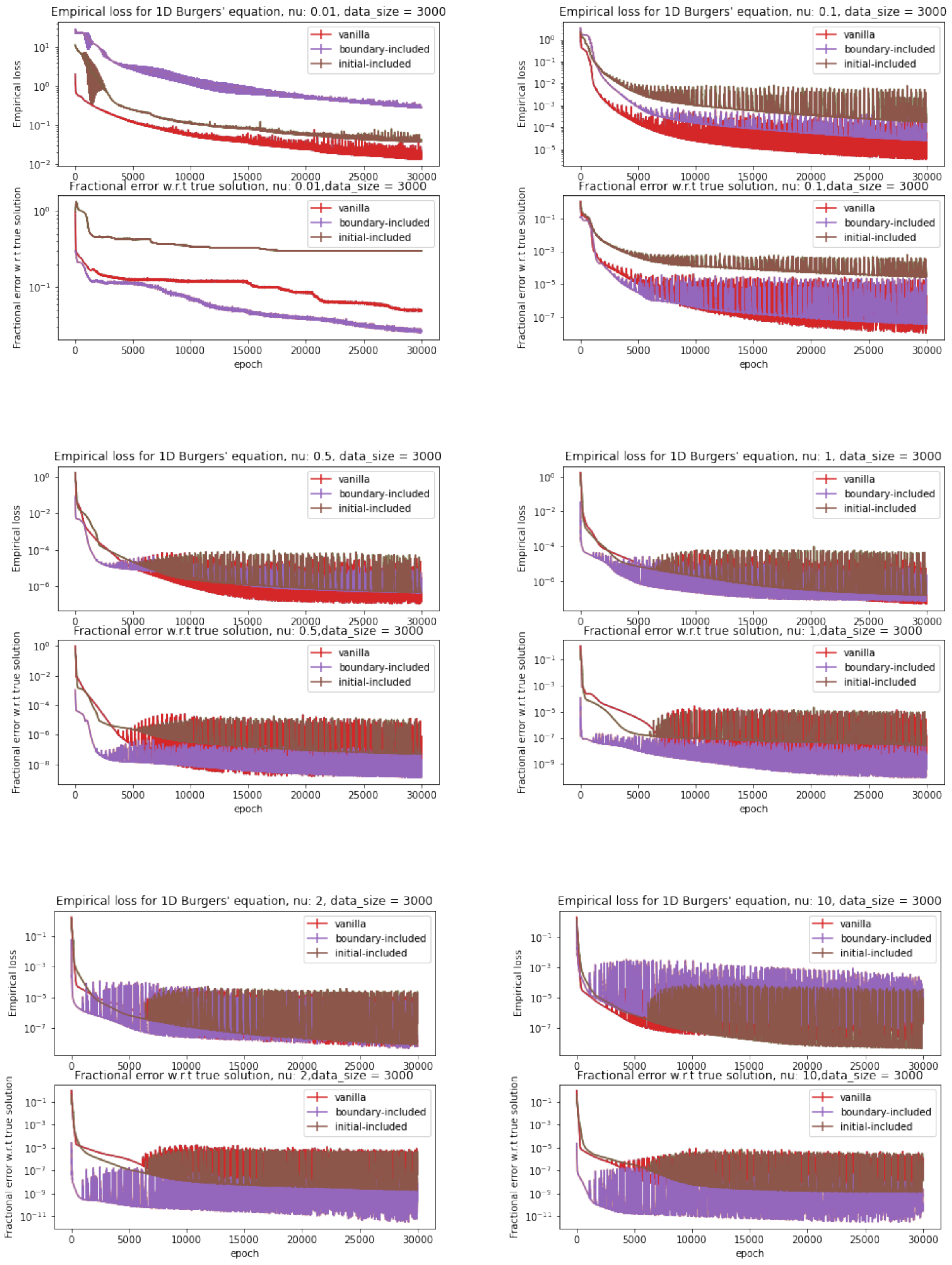


Figure 14: 1D Burger, epoch = 30000, para: 3441, lr = 0.0001,mesh size=3000

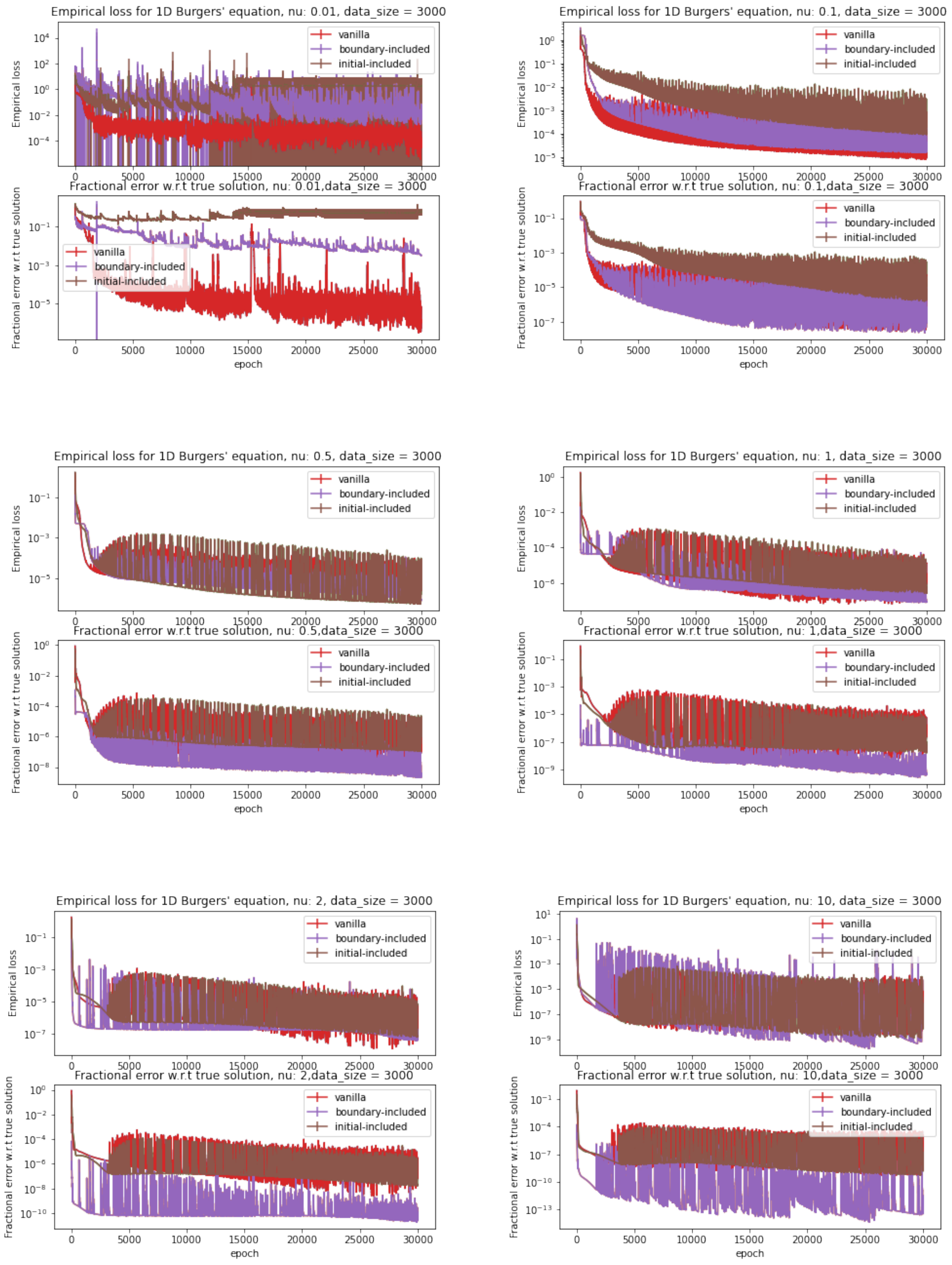


Figure 15: 1D Burger, epoch = 30000, para: 181801, lr = 0.0001,mesh size=3000

1D Burger, epoch =30000, lr=0.0001, mesh size=3000(Inside the brackets is error)			
ν	parameter=57	parameter=3441	parameter=181801
ν=0.01	Vanilla(1.0×10^{-1})	Vanilla(3.1×10^{-2})	Vanilla (5.2×10^{-7})
ν=0.1	Vanilla(1.1×10^{-7})	Vanilla(7.1×10^{-8})	Boundary-included(9.6×10^{-6})
ν=0.5	Vanilla(1.2×10^{-8})	Boundary-included(2.2×10^{-9})	Boundary-included(2.2×10^{-9})
ν=1	Boundary-included(7.5×10^{-10})	Boundary-included(7.2×10^{-10})	Boundary-included(4.2×10^{-10})
ν=2	Boundary-included(3.1×10^{-10})	Boundary-included(1.1×10^{-9})	Boundary-included(5.3×10^{-11})
ν=10	Boundary-included(1.0×10^{-12})	Boundary-included(1.3×10^{-10})	Boundary-included(7.1×10^{-13})

Table 8: Best model for 1D viscous Burger equation experiments with different values of ν and parameters

Table 8 shows the best models for different values of viscosity parameter ν on 1D viscous Burger equation with epoch=30000, lr=0.0001, and mesh size=3000 along with their respective fractional error.
Overall, the boundary-included model is the best performer, especially for higher values of ν and larger parameter values. However, for lower values of ν, the vanilla model performs almost equally well but fails to capture the details of the solution as effectively as the boundary-included model does. Therefore, it can be concluded that for 1D viscous Burger equation, the boundary-included model is the recommended model, especially when working with high values of ν or large parameter values.
Overall, from the results, we can get that the choice of model configuration has a significant impact on the performance of deep learning models in predicting solutions to the Burger equation.
In general, a model that is boundary-included performs better than the Vanilla model and initial-included model. This is likely because boundary conditions play an important role in determining the behavior of the solution to the 1D viscous Burger equation.

5.1.3 Time and device

Time here is the total time for 3 different models, and for each set of experiments, I repeated five times ("Time for 57" means the total time spent on the parameter equals 57).

1D viscid Burger, epoch =30000, lr=0.0001, mesh size=3000				
ν	Device	Time for 57	Time for 3441	Time for 181801
ν = 0.01	NVIDIA Tesla V100	173.3 mins	153.8 mins	233.9 mins
ν = 0.1	NVIDIA Tesla V100	163.1 mins	172.1 mins	221.7 mins
ν = 0.5	NVIDIA Tesla V100	164.3 mins	165.4 mins	237.1 mins
ν = 1	NVIDIA Tesla V100	144.9 mins	182.5 mins	228.9 mins
ν = 2	NVIDIA Tesla V100	153.6 mins	167.2 mins	218.3 mins
ν = 10	NVIDIA Tesla V100	172.8 mins	179.5 mins	211.2 mins

Table 9: Time and device for 1D viscid Burger equation experiments with different ν

5.2 Three-dimensional Viscid Burger Equation

The three-dimensional viscid Burger equation is a partial differential equation in fluid dynamics that describes the motion of an incompressible fluid. For here,d = 3, and the equation can be written as:

$$\partial_t \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \partial_t u_1 \\ \partial_t u_2 \\ \partial_t u_3 \end{bmatrix}, \ (\sum_{i=1}^{d} u_i \partial_{x_i}) \begin{bmatrix} u_1 \\ u_2 \\ u_3 \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{d} u_i \partial_{x_i} u_1 \\ \sum_{i=1}^{d} u_i \partial_{x_i} u_2 \\ \sum_{i=1}^{d} u_i \partial_{x_i} u_3 \end{bmatrix} = \begin{bmatrix} u_1\partial_{x_1}u_1 + u_2\partial_{x_2}u_1 + u_3\partial_{x_3}u_1 \\ u_1\partial_{x_1}u_2 + u_2\partial_{x_2}u_2 + u_3\partial_{x_3}u_2 \\ u_1\partial_{x_1}u_3 + u_2\partial_{x_2}u_3 + u_3\partial_{x_3}u_3 \end{bmatrix},$$

$$\nu \begin{bmatrix} \nabla^2 u_1 \\ \nabla^2 u_2 \\ \nabla^2 u_3 \end{bmatrix} = \nu \begin{bmatrix} \sum_{i=1}^{d} \partial_{x_i}^2 u_1 \\ \sum_{i=1}^{d} \partial_{x_i}^2 u_2 \\ \sum_{i=1}^{3} \partial_{x_i}^2 u_3 \end{bmatrix} = \nu \begin{bmatrix} \partial_{x_1}^2 u_1 + \partial_{x_2}^2 u_1 + \partial_{x_3}^2 u_1 \\ \partial_{x_1}^2 u_2 + \partial_{x_2}^2 u_2 + \partial_{x_3}^2 u_2 \\ \partial_{x_1}^2 u_3 + \partial_{x_2}^2 u_3 + \partial_{x_3}^2 u_3 \end{bmatrix}$$

The same as the 2D Burger equation, I will use u to represent u_1, v to represent u_2, and w to represent u_3.

$$\begin{cases} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} = \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} = \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}\right) \\ \frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} = \nu\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}\right) \end{cases}$$

From the (Shukla et al., 2016), we can get one of the exact solution for 3D viscid Burger equation:

$$\begin{cases} u(x,y,z,t) = -\frac{2}{\mathrm{Re}}\left(\frac{1+\cos(x)\sin(y)\sin(z)\exp(-t)}{1+x+\sin(x)\sin(y)\sin(z)\exp(-t)}\right), (x,y,z) \in \partial\Omega \\ v(x,y,z,t) = -\frac{2}{\mathrm{Re}}\left(\frac{\sin(x)\cos(y)\sin(z)\exp(-t)}{1+x+\sin(x)\sin(y)\sin(z)\exp(-t)}\right), (x,y,z) \in \partial\Omega \quad , t > 0 \\ w(x,y,z,t) = -\frac{2}{\mathrm{Re}}\left(\frac{\sin(x)\sin(y)\cos(z)\exp(-t)}{1+x+\sin(x)\sin(y)\sin(z)\exp(-t)}\right), (x,y,z) \in \partial\Omega \end{cases}$$

I made the decision to compare the precise answer to it in order to gauge how well my model predicted the bounds. I was able to get the starting and boundary conditions required to train and test my model by applying the precise solution at the boundary locations.
However, the same problem, I can not directly get the boundary condition and initial condition from this paper, and we know that using the exact solution in this way is often done in deep learning-based approaches to solving partial differential equations. We can choose the x, y, z, and t domains from 0 to 1.
Then we can get the initial condition for this Three-dimensional viscid Burger Equation experiment:

$$\begin{cases} u_0(x,y,z) = -\frac{2}{\mathrm{Re}}\left(\frac{1+\cos(x)\sin(y)\sin(z)}{1+x+\sin(x)\sin(y)\sin(z)}\right) \in \Omega \\ v_0(x,y,z) = -\frac{2}{\mathrm{Re}}\left(\frac{\sin(x)\cos(y)\sin(z)}{1+x+\sin(x)\sin(y)\sin(z)}\right) \in \Omega \\ w_0(x,y,z) = -\frac{2}{\mathrm{Re}}\left(\frac{\sin(x)\sin(y)\cos(z)}{1+x+\sin(x)\sin(y)\sin(z)}\right) \in \Omega \end{cases}$$

And for the boundary condition for this experiment:

$$\begin{cases} g_{x,0}(y,z,t) = -\frac{2}{\mathrm{Re}}\left(1+\sin(y)\sin(z)\exp(-t)\right) \\ g_{x,1}(y,z,t) = -\frac{2}{\mathrm{Re}}\left(\frac{1}{2+\sin(1)\sin(y)\sin(z)\exp(-t)}\right) \\ g_{y,0}(x,z,t) = -\frac{2}{\mathrm{Re}}\left(\frac{\sin(x)\sin(z)\exp(-t)}{1+x}\right) \\ g_{y,1}(x,z,t) = -\frac{2}{\mathrm{Re}}\left(\frac{\sin(x)\cos(1)\sin(z)\exp(-t)}{1+x+\sin(x)\sin(1)\sin(z)\exp(-t)}\right) \\ g_{z,0}(x,y,t) = -\frac{2}{\mathrm{Re}}\left(\frac{\sin(x)\sin(y)\exp(-t)}{1+x}\right) \\ g_{z,1}(x,y,t) = -\frac{2}{\mathrm{Re}}\left(\frac{\sin(x)\sin(y)\cos(1)\exp(-t)}{1+x+\sin(x)\sin(y)\sin(1)\exp(-t)}\right) \end{cases} , 1 > t > 0$$

where $\Omega = \{(x,y,z) : 0 \le x \le 1, 0 \le y \le 1, 0 \le z \le 1\}$ is the computational domain, where $g_{x,0}(y,z,t)$ and $g_{x,1}(y,z,t)$ is the boundary condition for u when $x = 0$ and $x = 1$, $g_{y,0}(x,z,t)$ and $g_{y,1}(x,z,t)$ is the boundary condition for v when $y = 0$ and $y = 1$, $g_{z,0}(x,y,t)$ and $g_{z,1}(x,y,t)$ is the boundary condition for w when $z = 0$ and $z = 1$. Also u_0, v_0 and w_0 is the initial condition. In the next section, I will try to prove the solution. $u\frac{\partial u}{\partial x}$ is the nonlinear convection term, $\frac{\partial u}{\partial t}$ is the unsteady term, $\nu = \frac{1}{\mathrm{Re}}$ is the kinematic viscosity.

5.2.1 Neural network for Three-dimensional Burger Equation

For the Three-dimensional Burger Equation, in my experiment, the net of this is just like: $N_3 : \mathbb{R}^4 \to \mathbb{R}^3$, $(x,y,z,t) \to N_{u,v,w}$, where (x,y,w,t) is the input and $N_{u,v,w}$ is the output. And the layer depth here is 5, so the details for this neural network is:

$$\mathcal{N}(\boldsymbol{x}) := \mathrm{A}_4(\sigma \mathrm{A}_3(\sigma \mathrm{A}_2(\sigma \mathrm{A}_1(\boldsymbol{x}))))$$

where$\mathrm{A}_1 : \mathbb{R}^4 \to \mathbb{R}^w$, $\mathrm{A}_2 : \mathbb{R}^w \to \mathbb{R}^w$, $\mathrm{A}_3 : \mathbb{R}^w \to \mathbb{R}^w$, $\mathrm{A}_4 : \mathbb{R}^w \to \mathbb{R}^3$, note: there are $2w^2 + 10w + 3$ parameters need to be trained in total in $\mathrm{N}(x)$

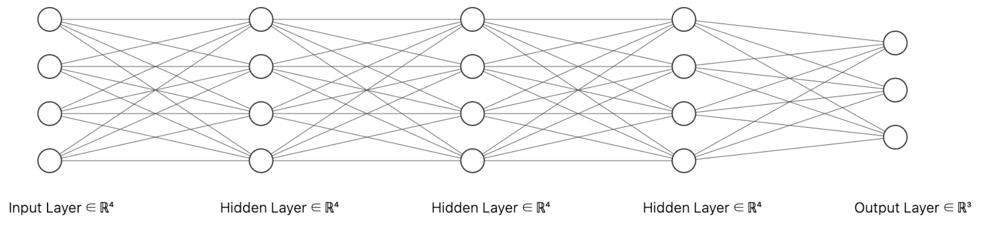


Figure 16: Neural network for Three-dimensional Burger equation when width=4

5.2.2 Model for Three-dimensional Viscid Burger Equation

Model for training on initial and boundary:

$$\mathrm{model}_{\mathrm{vanilla-u}}(x,y,z,t) \coloneqq \mathrm{N_u}(x,y,z,t)$$

$$\mathrm{model}_{\mathrm{vanilla-v}}(x,y,z,t) \coloneqq \mathrm{N_v}(x,y,z,t)$$

$$\mathrm{model}_{\mathrm{vanilla-w}}(x,y,z,t) \coloneqq \mathrm{N_w}(x,y,z,t)$$

Model for training on the boundary:

$$\mathrm{model}_{\mathrm{boundary-included-u}}(x,y,z,t) \coloneqq \mathrm{N_u}(x,y,z,t)\cdot x\cdot(1-x)+(1-x)\cdot g_{x,0}(y,z,t)+x\cdot g_{x,1}(y,z,t)$$

$$\mathrm{model}_{\mathrm{boundary-included-v}}(x,y,z,t) \coloneqq \mathrm{N_v}(x,y,z,t)\cdot y\cdot(1-y)+(1-y)\cdot g_{y,0}(x,z,t)+y\cdot g_{y,1}(x,z,t)$$

$$\mathrm{model}_{\mathrm{boundary-included-w}}(x,y,z,t) \coloneqq \mathrm{N_w}(x,y,z,t)\cdot z\cdot(1-z)+(1-z)\cdot g_{z,0}(x,y,t)+z\cdot g_{z,0}(x,y,t)$$

Model for training on the initial:

$$\mathrm{model}_{\mathrm{initial-included-u}}(x,y,z,t) \coloneqq \mathrm{N_u}(x,y,z,t)\cdot t+\cdot(1-t)\cdot u_0(x,y,z)$$

$$\mathrm{model}_{\mathrm{initial-included-v}}(x,y,z,t) \coloneqq \mathrm{N_v}(x,y,z,t)\cdot t+\cdot(1-t)\cdot v_0(x,y,z)$$

$$\mathrm{model}_{\mathrm{initial-included-w}}(x,y,z,t) \coloneqq \mathrm{N_w}(x,y,z,t)\cdot t+\cdot(1-t)\cdot w_0(x,y,z)$$

5.2.3 Loss for Three-dimensional Viscid Burger Equation

$\hat{\mathcal{R}}1$,$\hat{\mathcal{R}}2$ and $\hat{\mathcal{R}}3$ that are parts of the loss function correspond to the evaluations of differential operators.

$$\begin{cases} \hat{\mathcal{R}}_1 = \left\| \frac{\partial \hat{u}}{\partial t} + \hat{u}\frac{\partial \hat{u}}{\partial x} + \hat{v}\frac{\partial \hat{u}}{\partial y} + \hat{w}\frac{\partial \hat{u}}{\partial z} - \nu\left(\frac{\partial^2 \hat{u}}{\partial x^2} + \frac{\partial^2 \hat{u}}{\partial y^2} + \frac{\partial^2 \hat{u}}{\partial z^2}\right)\right\|^2_{[0,1]^3\times[0,1],\nu_1} \\ \hat{\mathcal{R}}_2 = \left\| \frac{\partial \hat{v}}{\partial t} + \hat{u}\frac{\partial \hat{v}}{\partial x} + \hat{v}\frac{\partial \hat{v}}{\partial y} + \hat{w}\frac{\partial \hat{v}}{\partial z} - \nu\left(\frac{\partial^2 \hat{v}}{\partial x^2} + \frac{\partial^2 \hat{v}}{\partial y^2} + \frac{\partial^2 \hat{v}}{\partial z^2}\right)\right\|^2_{[0,1]^3\times[0,1],\nu_1} \\ \hat{\mathcal{R}}_3 = \left\| \frac{\partial \hat{w}}{\partial t} + \hat{u}\frac{\partial \hat{w}}{\partial x} + \hat{v}\frac{\partial \hat{w}}{\partial y} + \hat{w}\frac{\partial \hat{w}}{\partial z} - \nu\left(\frac{\partial^2 \hat{w}}{\partial x^2} + \frac{\partial^2 \hat{w}}{\partial y^2} + \frac{\partial^2 \hat{w}}{\partial z^2}\right)\right\|^2_{[0,1]^3\times[0,1],\nu_1} \end{cases}$$

$\hat{\mathcal{R}}_4$,$\hat{\mathcal{R}}_5$ and $\hat{\mathcal{R}}_6$ correspond to the evaluation of boundary conditions(g_x, g_y and g_z are the boundary condition for the PDE).

$$\begin{cases} \hat{\mathcal{R}}_4 = \|\hat{u} - g_{x,0}(y,z,t)\|^2_{\{0\}\times[0,1]^2\times[0,1],\nu_2} + \|\hat{u} - g_{x,1}(y,z,t)\|^2_{\{1\}\times[0,1]^2\times[0,1],\nu_2} \\ \hat{\mathcal{R}}_5 = \|\hat{v} - g_{y,0}(x,z,t)\|^2_{[0,1]\times\{0\}\times[0,1]\times[0,1],\nu_2} + \|\hat{v} - g_{y,1}(x,z,t)\|^2_{[0,1]\times\{1\}\times[0,1]\times[0,1],\nu_2} \\ \hat{\mathcal{R}}_6 = \|\hat{w} - g_{z,0}(x,y,t)\|^2_{[0,1]^2\times\{0\}\times[0,1],\nu_2} + \|\hat{w} - g_{z,1}(x,y,t)\|^2_{[0,1]^2\times\{1\}\times[0,1],\nu_2} \end{cases}$$

$\hat{\mathcal{R}}_7$,$\hat{\mathcal{R}}_8$ and $\hat{\mathcal{R}}_9$ correspond to the evaluation of initial conditions(u_0, v_0 and w_0 are the initial condition for the PDE)

$$\begin{cases} \hat{\mathcal{R}}_7 = \|\hat{u} - u_0\|^2_{[0,1]^3,t=0,\nu_3} \\ \hat{\mathcal{R}}_8 = \|\hat{v} - v_0\|^2_{[0,1]^3,t=0,\nu_3} \\ \hat{\mathcal{R}}_9 = \|\hat{w} - w_0\|^2_{[0,1]^3,t=0,\nu_3} \end{cases}$$

where $\hat{u}$,$\hat{v}$ and $\hat{w}$ are approximate solutions predicted as an output of DNN.

Loss for training on initial and boundary:

$$\hat{\mathcal{R}}_{\text{vanilla}} = \left(\hat{\mathcal{R}}_1 + \hat{\mathcal{R}}_2 + \hat{\mathcal{R}}_3\right) + \left(\hat{\mathcal{R}}_4 + \hat{\mathcal{R}}_5 + \hat{\mathcal{R}}_6\right) + \left(\hat{\mathcal{R}}_7 + \hat{\mathcal{R}}_8 + \hat{\mathcal{R}}_9\right)$$

Loss for training on the boundary:

$$\hat{\mathcal{R}}_{\text{boundary-included}} = \left(\hat{\mathcal{R}}_1 + \hat{\mathcal{R}}_2 + \hat{\mathcal{R}}_3\right) + \left(\hat{\mathcal{R}}_7 + \hat{\mathcal{R}}_8 + \hat{\mathcal{R}}_9\right)$$

Loss for training on the initial:

$$\hat{\mathcal{R}}_{\text{initial-included}} = \left(\hat{\mathcal{R}}_1 + \hat{\mathcal{R}}_2 + \hat{\mathcal{R}}_3\right) + \left(\hat{\mathcal{R}}_4 + \hat{\mathcal{R}}_5 + \hat{\mathcal{R}}_6\right)$$

5.2.4 Same ν, different model

The two tables (Table 10 and Table 11) show the results of experiments conducted on the 3D viscid Burger equation with different values of ν and different models.
For the hyper-parameters here: a fixed number of epochs (10000), learning rate (0.001), mesh size (3000), and the width of the neural network (75, 3603 depending Based on the tables). we can design an experiment to investigate how changing the value of ν affects the performance of the model. To be noticed, for the boundary mesh size, it will contain 166 points for x = 0, 166 points for x = 1, 166 points for y = 0, 166 points for y = 0, 166 points for z = 0, and 170 points for z = 1.
I limited my analysis to two different neural network widths for the 3D Burger equation. The length of the training is the main factor. I used the CSF(The Computational Shared Facility) which the university provided (a v100 GPU), but compared to the 1D Burger equation, it takes a very long time for each different nu. So, to be a last resort, I greatly decreased the number of epochs while also choosing a smaller value for nu and narrowing the neural network's width.

3D Burger, epoch =10000, lr=0.001, parameter=75		
figure 13	ν	model
Experiment1	ν=0.01	**vanilla**:bulk mesh size=1000,initial mesh size=1000, boundary mesh size=1000 **boundary-included**: bulk mesh size = 1000,initial mesh size = 1000, boundary mesh size = 0, **initial-included**: bulk mesh size = 1000,initial mesh size = 0, boundary mesh size = 1000
Experiment2	ν=0.1	
Experiment3	ν=0.5	
Experiment4	ν=1	

Table 10: 3D Burger equation experiments with different values of ν when parameter=75

3D Burger, epoch =10000, lr=0.001, parameter=3603		
figure 13	ν	model
Experiment1	ν=0.01	**vanilla**:bulk mesh size=1000,initial mesh size=1000, boundary mesh size=1000 **boundary-included**: bulk mesh size = 1000,initial mesh size = 1000, boundary mesh size = 0, **initial-included**: bulk mesh size = 1000,initial mesh size = 0, boundary mesh size = 1000
Experiment2	ν=0.1	
Experiment3	ν=0.5	
Experiment4	ν=1	

Table 11: 3D Burger equation experiments with different values of ν when parameter=3603

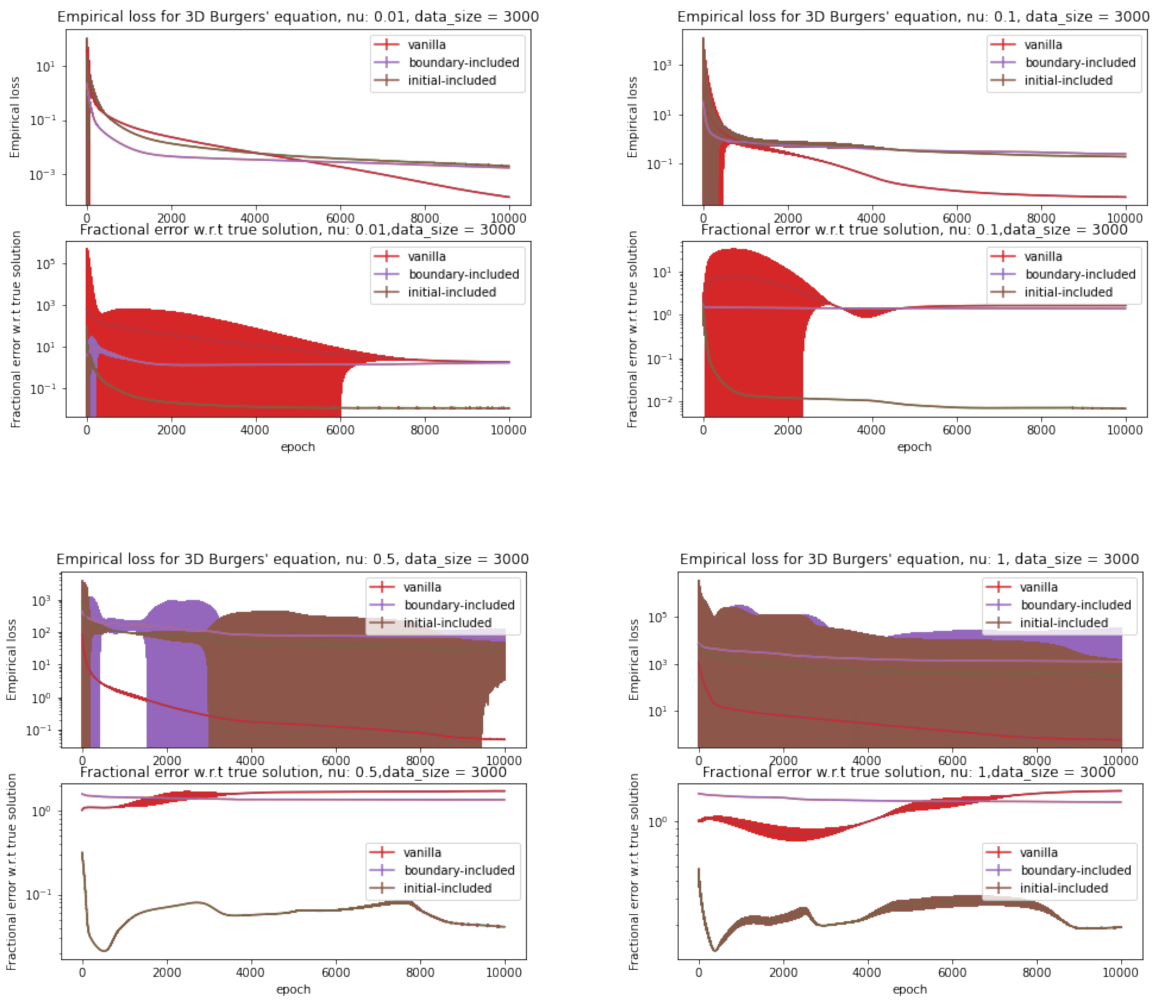


Figure 17: 3D Burger, epoch = 10000, para: 75, lr = 0.001,mesh size=3000, different ν

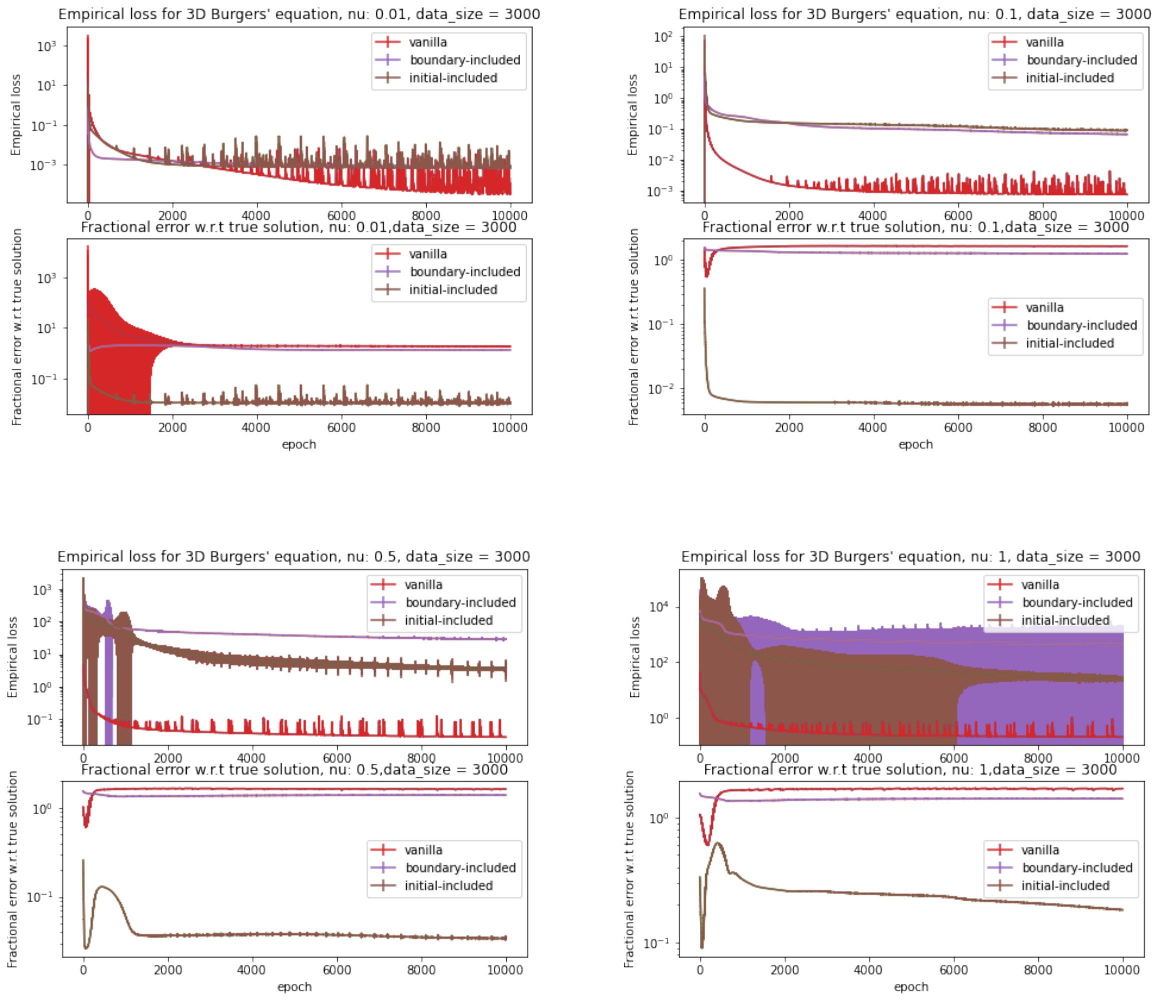


Figure 18: 3D Burger, epoch = 10000, para: 3603, lr = 0.001,mesh size=3000, different ν

3D Burger, epoch =10000, lr=0.001(Inside the brackets is error)		
ν	parameter=75	parameter=3603
ν=0.01	Initial-included(1.1×10^{-2})	Initial-included(9.8×10^{-3})
ν=0.1	Initial-included(6×10^{-3})	Initial-included(5.1×10^{-3})
ν=0.5	Initial-included(4×10^{-2})	Initial-included(3.4×10^{-2})
ν=1	Initial-included(1.9×10^{-1})	Initial-included(1.8×10^{-1})

Table 12: Best model for 3D viscous Burger equation experiments with different values of ν and parameters

From the two tables, we can observe that as the viscosity coefficient ν increases, the fractional error will increase, which indicates a decrease in model performance. This trend is consistent for both values of the parameter tested.
For the 3D viscous Burger equation, different from 1D viscous Burger equation, *we can see that the initial-included model configuration consistently has the best performance than the other two models(vanilla and boundary-included). This observation indicates that including the initial conditions in the training, data can*

improve the accuracy of the model predictions.
Moreover, we can see that the error will decrease when the number of the parameter is from 75 to 3603 for all values of ν. However, the progress is not very significant and the result suggests that the performance of the model is not very sensitive to changes in the width of the neural network.

5.2.5 Time and device

Time here is the total time for 3 different models, and for each set of experiments, I repeated three times.

3D viscid Burger, epoch =3000, lr=0.001, mesh size=3000			
ν	Device	Time for 75	Time for 3603
$\nu = 0.01$	NVIDIA Tesla V100	382.3 mins	1351.9 mins
$\nu = 0.1$	NVIDIA Tesla V100	402.9 mins	1264.6 mins
$\nu = 0.5$	NVIDIA Tesla V100	392.4 mins	1295.1 mins
$\nu = 1$	NVIDIA Tesla V100	398.0 mins	1311.6 mins

Table 13: Time and device for 3D viscid Burger equation experiments with different ν

5.3 Conclusion for Viscid Burger Equation

From Table 8 and Table 12, *for the 1D viscid Burger equation and 3D viscid Burger equation, the best models are boundary-included and initial-included respectively.*
For the 3D viscous Burger equation, the initial-included model may work well because it prioritizes an accurate representation of the initial state of the system. This can be important for systems that are highly dependent on the initial conditions or where the initial conditions have a significant impact on the evolution of the system. Additionally, in a 3D system with many degrees of freedom, accurately representing the initial conditions can help reduce the error accumulation during simulation.
On the other hand, for the 1D viscous Burger equation, the boundary-included model may work better because this equation depends heavily on boundary conditions. Accurately representing the boundary conditions can be more important than the initial conditions for this type of system. Additionally, simulations for one-dimensional systems may not require as much focus on the initial conditions as multi-dimensional systems because the dynamics of 1D systems are relatively simpler.

I have one other finding, *for the 1D viscous Burger equation, as the viscosity increases, the accuracy increases. But for the 3D viscous Burger equation, when the viscosity increases, the accuracy will decrease.*
For the 1D viscous Burger equation, increasing viscosity can help reduce numerical errors and improve accuracy because it smooths out the solution and reduces the impact of small-scale fluctuations. This smoothing effect is particularly noticeable when viscosity is low, and the solution is prone to high-frequency oscillations.
However, for the 3D viscous Burger equation, due to the limited experiments, the selected hyperparameters are not as many as 1D, so I can not explain why the accuracy will decrease when the viscosity increases. The relationship between viscosity and accuracy is more complex here

6 Experiment on "Double Decent"

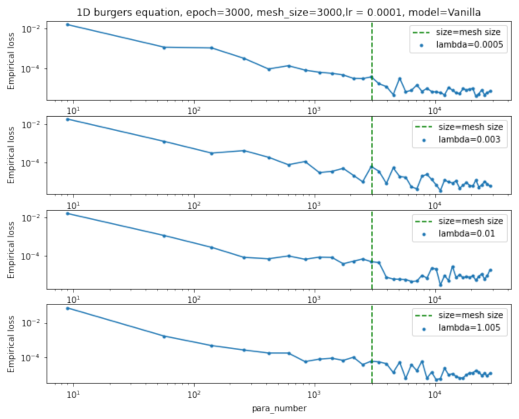
In order to ascertain whether the "Double Descent" phenomenon occurs in the standard PINNs PDE-solving problem, the experiments in this section primarily focused on the One-dimensional Inviscid Burger Equation and Two-dimensional Inviscid Burger Equation with the same configuration as we did in section 4.1 and section 4.2. Furthermore, rather than regularising the weight decay which(Nakkiran et al., 2021) did for mitigating, we applied a weight λ on the penalty terms in the loss function to demonstrate how the lambda weight can affect performance as the size of the neural net grows larger. The boundary-included model is also considered in this experiment to create a comparison group.
As a result of including the factor λ in the loss function.
Loss for training on initial and boundary:

$$\hat{\mathcal{R}}_{\text{vanilla}}(\hat{u}) = \left\| \frac{\partial \hat{u}}{\partial t} + \hat{u}\frac{\partial \hat{u}}{\partial x} \right\|^2_{[0,1]\times[0,1],\nu_1} + \lambda \|\hat{u} - F(x)\|^2_{t=0,[0,1],\nu_2} + \lambda \|\hat{u} - g(x,t)\|^2_{\{0,1\}\times[0,1],\nu_3}$$

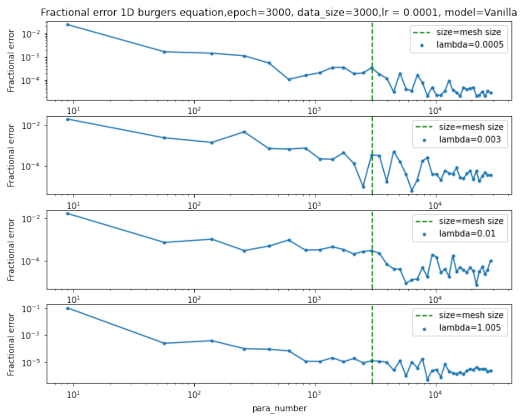
Loss for training on the boundary:

$$\hat{\mathcal{R}}_{\text{boundary-included}}(\hat{u}) = \left\| \frac{\partial \hat{u}}{\partial t} + \hat{u}\frac{\partial \hat{u}}{\partial x} \right\|^2_{[0,1]\times[0,1],\nu_1} + \lambda \|\hat{u} - F(x)\|^2_{t=0,[0,1],\nu_2}$$

The maximum size neural network employed in this experiment consisted of approximately 30000 parameters, while the total number of training points was 3000 (each 1000 from the bulk, boundary, and initial condition). The figures below can be divided into left and right parts according to the dotted line in the vertical direction. The part on the left is the bias-variance trade-off in traditional statistical learning. At this time, the parameters of the model are less than the interpolation threshold. In fact, the threshold is the point where the number of models is equal to the number of samples. And the epoch here is 3000 for each different parameter. We also tested regularisation parameter *lambda* values ranging from zero to over one, including 0.0005,0.003,0.01, and 1.05.

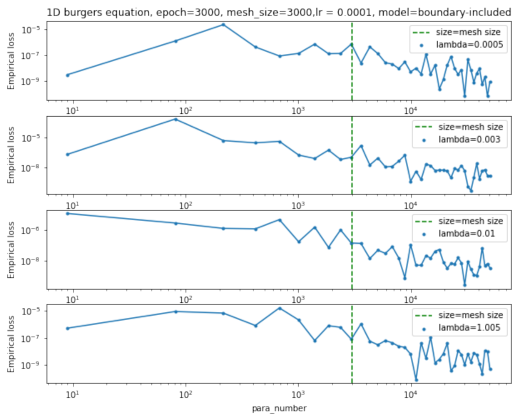


(a) Empirical loss for 1D Burger Equation for different λ

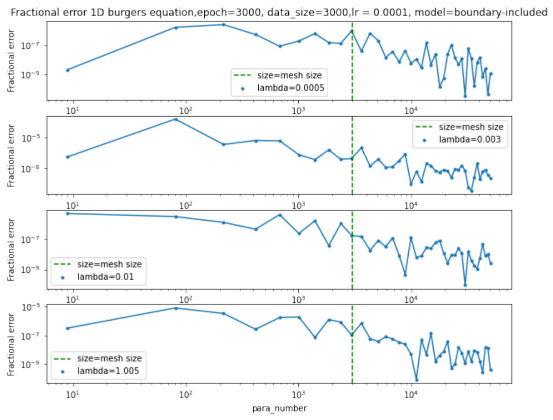


(b) Fractional error 1D Burger Equation for different λ

Figure 19: 1D Burger equation for different λ,model=Vanilla



(a) Empirical loss for 1D Burger Equation for different λ



(b) Fractional error 1D Burger Equation for different λ

Figure 20: 1D Burger equation for different λ,model=Boundary-included

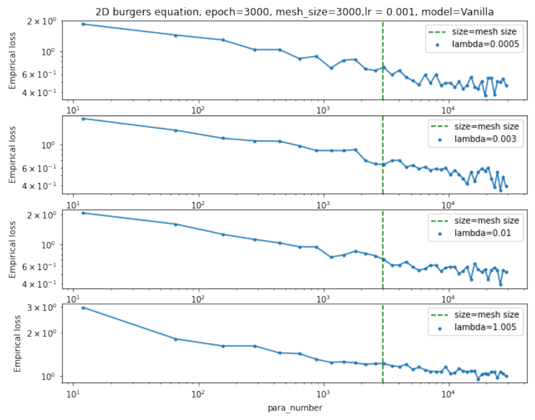
In Figure 19 and Figure 20, no matter what the lambda is, it is difficult to determine where the second descent originates. In Figure 19, the loss and error gradually slow decline just like the traditional statistical learning. In Figure 20, even though we can see one peak around the green line, which is where the double descent highest peak should be (Nakkiran et al., 2021), it is difficult to explain why the double descent is there.
For the loss function for the 2D Burger Equation, we can see the section 4.2.4, and then I will add the λ in the loss. So the new loss function will show below.
Loss for training on initial and boundary:

$$\hat{\mathcal{R}}_{\text{vanilla}} = \left(\hat{\mathcal{R}}_1 + \hat{\mathcal{R}}_2\right) + \lambda\left(\hat{\mathcal{R}}_3 + \hat{\mathcal{R}}_4\right) + \lambda\left(\hat{\mathcal{R}}_5 + \hat{\mathcal{R}}_6\right)$$

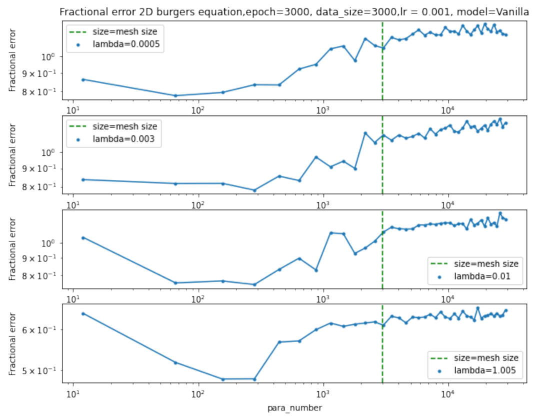
Loss for training on the boundary:

$$\hat{\mathcal{R}}_{\text{boundary-included}} = \left(\hat{\mathcal{R}}_1 + \hat{\mathcal{R}}_2\right) + \lambda\left(\hat{\mathcal{R}}_5 + \hat{\mathcal{R}}_6\right)$$

The configuration is consistent with the last experiment.

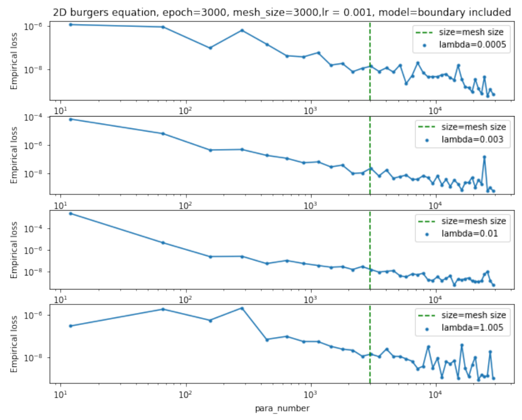


(a) Empirical loss for 2D Burger Equation for different λ

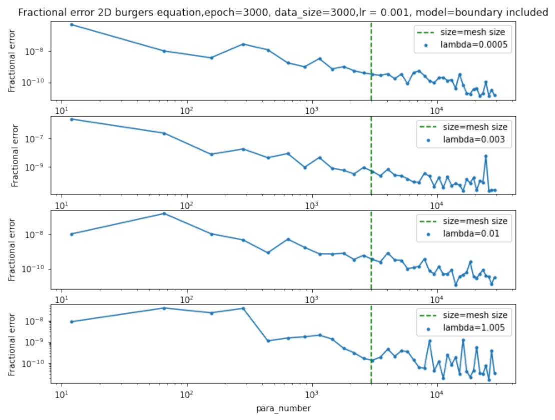


(b) Fractional error 2D Burger Equation for different λ

Figure 21: 2D Burger equation for different λ,model=Vanilla



(a) Empirical loss for 2D Burger Equation for different λ



(b) Fractional error 2D Burger Equation for different λ

Figure 22: 2D Burger equation for different λ,model=Boundary-included

In Figure 21 and Figure 22, no matter what the lambda is, it is also difficult to determine where the second descent originates. In Figure 21, the error even gradually slowly increase which is very strange. In Figure 22, the loss and error gradually slow decline just like traditional statistical learning.

While it may not be fair to say unequivocally that "Double Descent" effects exist in PINN models regardless of whether previous information is included, our studies do reveal that increasing neural network size does not necessarily result in fractional error reductions.

7 Inverse Problem

The inverse problem for the Burger equation involves determining the initial and/or boundary conditions that give rise to a particular velocity field as a solution of the equation. In other words, given a desired velocity field at some later time, the goal is to determine what the velocity field must have been initially and/or at the boundaries in order to evolve into this state.
This problem is difficult because the Burger equation is nonlinear and does not have an explicit solution in general. Moreover, the initial and boundary conditions are often unknown or poorly defined in practice, making it challenging to formulate and solve the inverse problem. So in my experiment, I will try to discuss the inverse problem for the 1D Burger equation.

7.1 1D Burger equation for inverse problem

The given equation is a one-dimensional convection-diffusion equation, which describes the evolution of a velocity field u in a fluid under the influence of both convection and diffusion. The equation is given by:

$$\frac{\partial u}{\partial t} + \lambda_1 u \frac{\partial u}{\partial x} = \lambda_2 \frac{\partial^2 u}{\partial x^2} x \in [-1,1] t \in [0,1]$$

where λ_1 and λ_2 are constants that represent the strength of the convection and diffusion terms, respectively. The domain of the problem is $x \in [-1,1]$ and $t \in [0,1]$. This means that the velocity field is defined on a finite interval in space and time.

Forward Problem: Model → Data (Predict)
Inverse Problem: Data → Model (i.e., actually, we get the Model's parameters)
Data → PINN → Model Parameters (i.e., our PDE parameters). For parameterized and nonlinear partial differential equations of the general form (Raissi et al., 2017):

$$u_t + \mathcal{N}[u;\lambda] = 0$$

where, $u(x,t)$ is the hidden solution and $\mathcal{N}[;\lambda]$ is a nonlinear operator parameterized by λ.
In short: We will use a PINN to get λ .

7.2 Analysis

Let:

$$\begin{aligned} u_t &= \frac{\partial u}{\partial t} \\ u_x &= \frac{\partial u}{\partial x} \\ u_{xx} &= \frac{\partial^2 u}{\partial x^2} \\ \mathcal{N}[u;\lambda] &= \lambda_1 u u_x - \lambda_2 u_{xx} \end{aligned}$$

Our PDE is described as:

$u_t + \lambda_1 u u_x = \lambda_2 u_{xx}$

If we rearrange our PDE, we get:

$u_t + \lambda_1 u u_x - \lambda_2 u_{xx} = 0$

Or,

$u_t + \mathcal{N}[u;\lambda] = 0$

So we can use a PINN to obtain λ ! In our case (from the reference solution) $\lambda = [\lambda_1, \lambda_2] = [1, \nu] = \left[1, \frac{1}{100\pi}\right]$

7.3 Neural Network

A Neural Network is a function (Raissi et al., 2019a):

$\mathcal{N}(X) = W_n\sigma_{n-1}\left(W_{n-1}\sigma_{n-2}\left(\ldots\left(W_2\left(W_1X + b_1\right) + b_2\right) + ..\right) + b_{n-1}\right) + b_n$

Note: We usually train our N by iteratively minimizing a loss function in the training dataset(known data) to get W_i and b_i .

7.4 PINNs=Neural Network + PDE

We can use a neural network to approximate any function (Universal Approximation Theorem(Lu et al., 2021)):

$$N(x,t) \approx u(x,t)$$

Since $N(x,t)$ is a function, we can obtain its derivatives: $\frac{\partial N}{\partial t}, \frac{\partial^2 N}{\partial x^2}$, etc.. (Automatic Diferentiation) Assume:

$$N(t,x) \approx u(t,x)$$

Then:

$$N_t + \lambda_1 N N_x - \lambda_2 N_{xx} \approx u_t + uu_x - \lambda u_{xx} = 0$$

And:

$$N_t + \lambda_1 N N_x - \lambda_2 N_{xx} \approx 0$$

We define this function as f :

$$f(t,x) = N_t + \lambda_1 N N_x - \lambda_2 N_{xx}$$

Remember our operator:

$$f(t,x) = N_t + \mathcal{N}[N,\lambda]$$

So:

$$f(t,x) \approx 0$$

7.5 Define Loss function

We evaluate f in a certain number of points (N_u) inside our domain (x,t). Then we iteratively minimize a loss function related to f :

$$MSE_f = \frac{1}{N_u}\sum_{i=1}^{N_u}\left|f\left(t_u^i, x_u^i\right)\right|^2$$

Usually, the training data set is a set of points from which we know the answer. In our case, points inside our domain (i.e., interior points). Remember that this is an inverse problem, so we know the data. Since we know the outcome, we select N and compute the MSE_u^{**} (compare it to the reference solution).

$$MSE_u = \frac{1}{N_u}\sum_{i=1}^{N_u}\left|u\left(t_u^i, x_u^i\right) - N\left(t_u^i, x_u^i\right)\right|^2$$

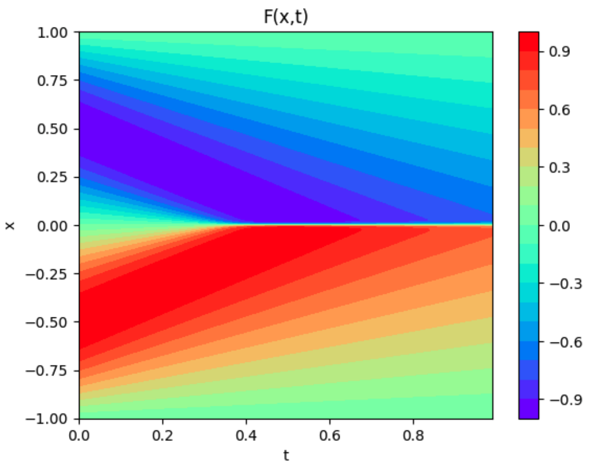
Please note that $\left\{t_u^i, x_u^i\right\}_{i=1}^{N}$ are the same in MSE_f and MSE_u
Total Loss:

$$MSE = MSE_u + MSE_f$$

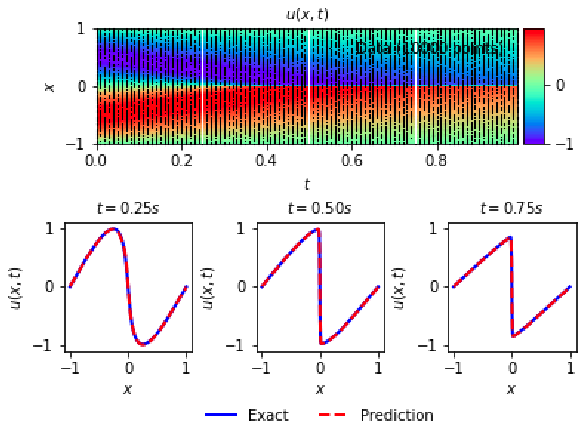
NOTE: We minimize MSE to obtain the N' 's parameters (i.e, W_i and b_i) and the ODE parameters (i.e., λ) $\rightarrow$ We will ask our neural network to find our λ .

7.6 Visualize data

Figure 23(a) is the exact value of my example, which I drew through a .mat file. The upper part of Figure 23(b) is the prediction picture obtained through training, and the lower part is the cut-away picture at different times. It can be pleasantly surprising to find that for the 1D Burger, the effect is very good.



(a) Exact solution for 1D inverse Burger Equation



(b) Predict solution for 1D inverse Burger Equation

Figure 23: Visualize data for 1D inverse Burger Equation

8 Summary and Outlook

8.1 Project Work Summary

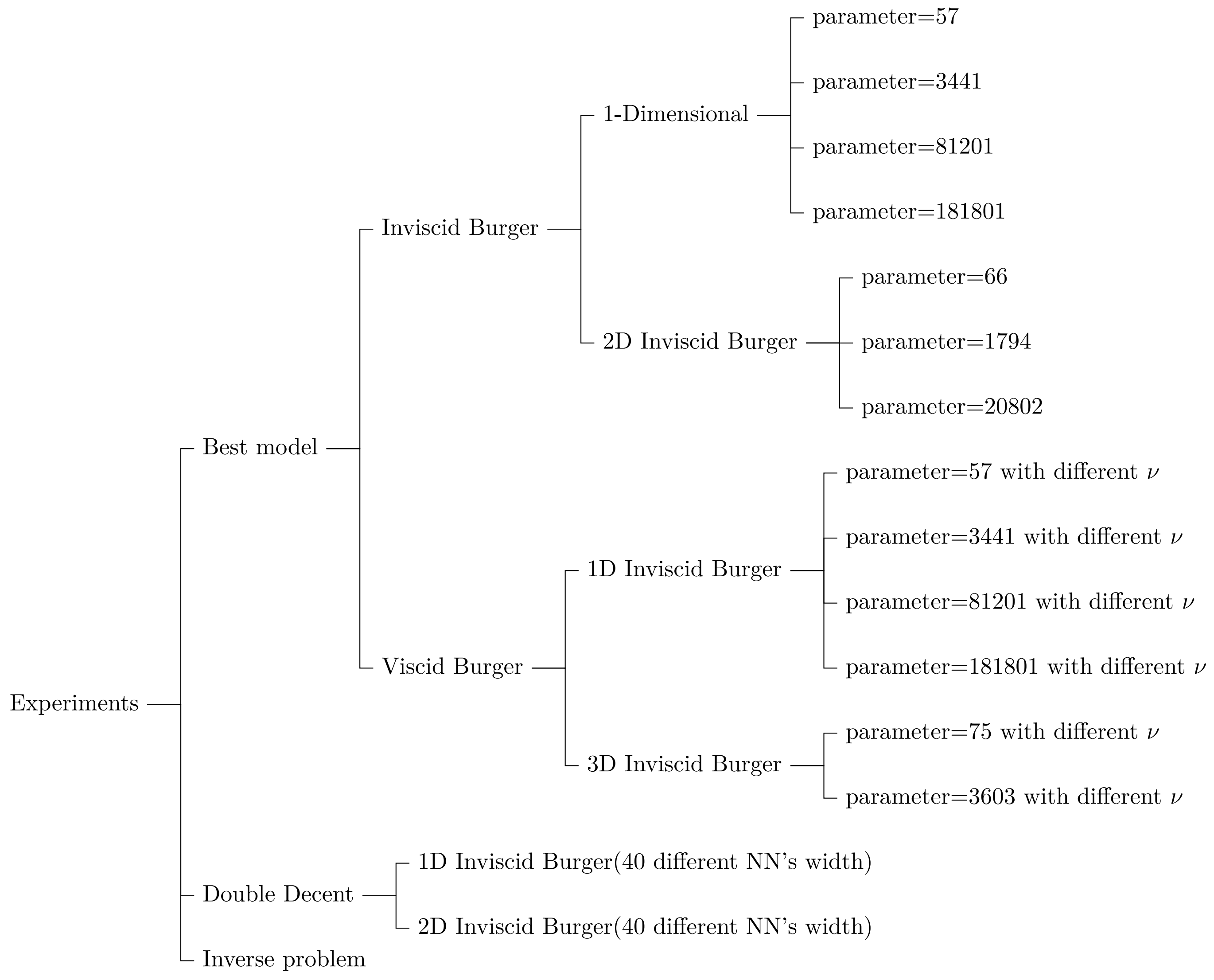
The main goal of this project is to use Physics-Informed Neural Networks to solve PDEs and improve the models. Previously, I noted that PINN can not always guarantee to provide a better solution. By using different models which add the boundary or initial conditions, I find that 1D, 2D, and 3D Burger Equations have improved their performance to some degree. However, from these experiments, I can not give a definitive conclusion about which model will always have the best performance. For 1D and 2D Burger Equations, the Boundary-included model will give me the best solution, meanwhile, the Initial-included model can provide the most accurate prediction for 3D Burger Equations.

Burger Equation does not have the Double Decent phenomenon. In contrast to deep learning models, maybe the 1D and 2D Burger Equations are relatively simple models with a small number of parameters. It has been shown that the equation does not exhibit the double descent phenomenon because there are no spurious features that can cause overfitting or underestimation bias. The simplicity of the Burger Equation means that it is less prone to overfitting or underfitting, and its performance does not degrade as the number of parameters increases.

I also discuss a very challenging part which is the inverse problem. However, there is no in-depth research, just a simple analysis of the inverse problem of the 1D Burger Equation with reference to other people's papers. Through visualization, I found that the effect of this experiment is actually good. However, this is a very difficult field, and this project is only briefly introduced and may continue to be explored in subsequent research.

The following dendrogram can represent my main experiments, for the part of the "Best model", the total number is 4*3 + 3*3 + 4*6*3 + 2*4*3 = 117. For the part of the "Double Decent", the total number is 40 + 40 = 80. for the part about the "Inverse problem", I only do it once. So the total experiment times are 198.

In the end, I counted the total training time and found that it exceeded 200 hours, 1D Burger took about 50 hours, 2D Burger took about 70 hours, 3D Burger took nearly 80 hours, and 20 hours for the Double Decent experiment. so I am very grateful for the CSF provided by the university.



Experiments
Best model
Inviscid Burger
1-Dimensional
parameter=57
parameter=3441
parameter=81201
parameter=181801
2D Inviscid Burger
parameter=66
parameter=1794
parameter=20802
Viscid Burger
1D Inviscid Burger
parameter=57 with different ν
parameter=3441 with different ν
parameter=81201 with different ν
parameter=181801 with different ν
3D Inviscid Burger
parameter=75 with different ν
parameter=3603 with different ν
Double Decent
1D Inviscid Burger(40 different NN's width)
2D Inviscid Burger(40 different NN's width)
Inverse problem

8.2 Possible future developments

Follow-up may continue to study higher-dimensional Burger Equations to see if our model can still improve accuracy. Similarly, for high-dimensional equations, the parameters mean more, so it is also convenient for us to observe the phenomenon of Double Decent. In recent years, I have found that more and more people are studying the inverse problem, and some excellent papers (Apraiz et al., 2022),(Gamzaev, 2017) have been produced. Overall, exploring the inverse problem in PDEs is a challenging but important area of research that has wide-ranging applications and can lead to significant advancements in many fields.

From a larger perspective, the Kepler paradigm driven by data, and the Newton paradigm driven by first principles are the two basic paradigms of scientific research for hundreds of years. The popularity of the current "AI for Science" trend may promote the deep integration of the two existing paradigms and stimulate a new scientific revolution.
At present, AI for Science has made notable breakthroughs, but the results are mainly point-like, including molecular biology, quantum mechanics, etc., which have not yet formed large-scale results, and there is still

a lot of room for breakthroughs.
Therefore, we need to form computational thinking to carry out scientific exploration, and at the same time, we can create a new future of science by properly applying the tool of artificial intelligence. AI for Science is a trend of artificial intelligence full of bright future.

References

Ravi P Agarwal. Burgers' equation (viscous). *Indian Institute of Space Science and Technology*, 2011.

Jone Apraiz, Anna Doubova, Enrique Fernández-Cara, and Masahiro Yamamoto. Some inverse problems for the burgers equation and related systems. *Communications in Nonlinear Science and Numerical Simulation*, 107:106113, 2022.

Christian Beck, Martin Hutzenthaler, Arnulf Jentzen, and Benno Kuckuck. An overview on deep learning-based approximation methods for partial differential equations. *arXiv preprint arXiv:2012.12348*, 2020.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

Léon Bottou. Stochastic gradient descent tricks. *Neural Networks: Tricks of the Trade: Second Edition*, pp. 421–436, 2012.

Jingrun Chen, Rui Du, and Keke Wu. A comprehensive study of boundary conditions when solving pdes by dnns. *arXiv preprint arXiv:2005.04554*, 2020.

Antonia Creswell, Tom White, Vincent Dumoulin, Kai Arulkumaran, Biswa Sengupta, and Anil A Bharath. Generative adversarial networks: An overview. *IEEE signal processing magazine*, 35(1):53–65, 2018.

Salvatore Cuomo, Vincenzo Schiano Di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics–informed neural networks: where we are and what's next. *Journal of Scientific Computing*, 92(3):88, 2022.

Hans Fangohr. Pdes and numerical methods. https://www.southampton.ac.uk/~fangohr/teaching/comp6024/comp6024-pdes.pdf, 2016.

Carlos A Felippa. Introduction to finite element methods. *University of Colorado*, 885, 2004.

Kh M Gamzaev. Numerical solution of combined inverse problem for generalized burgers equation. *Journal of Mathematical Sciences*, 221(6):833–840, 2017.

Christian Glusa and Enrique Otárola. Error estimates for the optimal control of a parabolic fractional pde. *SIAM Journal on Numerical Analysis*, 59(2):1140–1165, 2021.

Philip Hartman. *Ordinary differential equations.* SIAM, 2002.

Trevor Hastie, Andrea Montanari, Saharon Rosset, and Ryan J Tibshirani. Surprises in high-dimensional ridgeless least squares interpolation. *The Annals of Statistics*, 50(2):949–986, 2022.

Paul G Huray. *Maxwell's equations.* John Wiley & Sons, 2011.

Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

Aditi Krishnapriyan, Amir Gholami, Shandian Zhe, Robert Kirby, and Michael W Mahoney. Characterizing possible failure modes in physics-informed neural networks. *Advances in Neural Information Processing Systems*, 34:26548–26560, 2021.

Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via deeponet based on the universal approximation theorem of operators. *Nature machine intelligence*, 3(3):218–229, 2021.

Masaharu Matsumoto. Application of deep galerkin method to solve compressible navier-stokes equations. *TRANSACTIONS OF THE JAPAN SOCIETY FOR AERONAUTICAL AND SPACE SCIENCES*, 64 (6):348–357, 2021.

Dagmar Medková. The laplace equation. *Boundary value problems on bounded and unbounded Lipschitz domains. Springer, Cham*, 2018.

Xuhui Meng and George Em Karniadakis. A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse pde problems. *Journal of Computational Physics*, 401: 109020, 2020.

RC Mittal and Poonam Singhal. Numerical solution of burger's equation. *Communications in numerical methods in engineering*, 9(5):397–406, 1993.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *Journal of Statistical Mechanics: Theory and Experiment*, 2021(12):124003, 2021.

Garbled Notes. Differences between l1 and l2 as loss function and regularization. *online] https://web. archive. org/web/20140328055117/http://www. chioka. in/differences-between-11-and-12-as-loss-function-and-regularization*, 2014.

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, 2019a. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part ii): Data-driven discovery of nonlinear partial differential equations, 2017.

Maziar Raissi, Paris Perdikaris, and George 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, 2019b.

Prajit Ramachandran, Barret Zoph, and Quoc V. Le. Searching for activation functions, 2017.

Deep Ray, Orazio Pinti, and Assad A. Oberai. Deep learning and computational physics (lecture notes), 2023.

Michael Renardy and Robert C Rogers. *An introduction to partial differential equations*, volume 13. Springer Science & Business Media, 2006.

A Salih. Inviscid burgers' equation. *A university lecture, Indian Institute of Space Science and Technology*, 2015.

Simon Schneiderbauer and Michael Krieger. What do the navier–stokes equations mean? *European Journal of Physics*, 35(1):015020, dec 2013. doi: 10.1088/0143-0807/35/1/015020. URL https://dx.doi.org/10.1088/0143-0807/35/1/015020.

Zhen-Su She, Erik Aurell, and Uriel Frisch. The inviscid burgers equation with initial data of brownian type. *Communications in mathematical physics*, 148:623–641, 1992.

HS Shukla, Mohammad Tamsir, Vineet K Srivastava, and Mohammad Mehdi Rashidi. Modified cubic b-spline differential quadrature method for numerical solution of three-dimensional coupled viscous burger equation. *Modern Physics Letters B*, 30(11):1650110, 2016.

Stephen G Simpson. Which set existence axioms are needed to prove the cauchy/peano theorem for ordinary differential equations? *The Journal of symbolic logic*, 49(3):783–802, 1984.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, 2018. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.08.029. URL https://www.sciencedirect.com/science/article/pii/S0021999118305527.

Hind Taud and JF Mas. Multilayer perceptron (mlp). *Geomatic approaches for modeling land change scenarios*, pp. 451–455, 2018.

James William Thomas. *Numerical partial differential equations: finite difference methods*, volume 22. Springer Science & Business Media, 2013.

C. Y. Wang. Exact solutions of the steady-state navier-stokes equations. *Annual Review of Fluid Mechanics*, 23(1):159–177, 1991. doi: 10.1146/annurev.fl.23.010191.001111. URL https://doi.org/10.1146/annurev.fl.23.010191.001111.

Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021.

E Weinan, Jiequn Han, and Arnulf Jentzen. Algorithms for solving high dimensional pdes: from nonlinear monte carlo to machine learning. *Nonlinearity*, 35(1):278, 2021.

Bing Yu et al. The deep ritz method: a deep learning-based numerical algorithm for solving variational problems. *Communications in Mathematics and Statistics*, 6(1):1–12, 2018.

Hongqing Zhu, Huazhong Shu, and Meiyu Ding. Numerical solutions of two-dimensional burgers' equations by discrete adomian decomposition method. *Computers & Mathematics with Applications*, 60(3):840–848, 2010.

A Some Notation

Sets and Graphs

$\mathbb{A}$	A set
$\mathbb{R}$	The set of real numbers
$\{0, 1\}$	The set containing 0 and 1
$\{0, 1, \dots, n\}$	The set of all integers between 0 and n
$[a, b]$	The real interval including a and b
$(a, b]$	The real interval excluding a but including b
$\mathbb{A} \backslash \mathbb{B}$	Set subtraction, i.e., the set containing the elements of $\mathbb{A}$ that are not in $\mathbb{B}$
$\mathcal{G}$	A graph
$Pa_{\mathcal{G}}(\mathrm{x}_i)$	The parents of x_i in $\mathcal{G}$

Numbers and Arrays

a	A scalar (integer or real)
$\boldsymbol{a}$	A vector
$\boldsymbol{A}$	A matrix
$\mathsf{\mathbf{A}}$	A tensor
$\boldsymbol{I}_n$	Identity matrix with n rows and n columns
$\boldsymbol{I}$	Identity matrix with dimensionality implied by context
$\boldsymbol{e}^{(i)}$	Standard basis vector $[0, \dots, 0, 1, 0, \dots, 0]$ with a 1 at position i
$\text{diag}(\boldsymbol{a})$	A square, diagonal matrix with diagonal entries given by $\boldsymbol{a}$
a	A scalar random variable
$\mathbf{a}$	A vector-valued random variable
$\mathbf{A}$	A matrix-valued random variable

Indexing

a_i	Element i of vector $\boldsymbol{a}$, with indexing starting at 1
a_{-i}	All elements of vector $\boldsymbol{a}$ except for element i
$A_{i,j}$	Element i, j of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{i,:}$	Row i of matrix $\boldsymbol{A}$
$\boldsymbol{A}_{:,i}$	Column i of matrix $\boldsymbol{A}$
$\mathsf{A}_{i,j,k}$	Element (i, j, k) of a 3-D tensor $\mathsf{\mathbf{A}}$
$\mathsf{\mathbf{A}}_{:,:,i}$	2-D slice of a 3-D tensor
a_i	Element i of the random vector $\mathbf{a}$

Probability and Information Theory

$P(\mathrm{a})$	A probability distribution over a discrete variable
$p(\mathrm{a})$	A probability distribution over a continuous variable, or over a variable whose type has not been specified
$\mathrm{a} \sim P$	Random variable a has distribution P
$\mathbb{E}_{\mathrm{x}\sim P}[f(x)]$ or $\mathbb{E}f(x)$	Expectation of $f(x)$ with respect to $P(\mathrm{x})$
$\mathrm{Var}(f(x))$	Variance of $f(x)$ under $P(\mathrm{x})$
$\mathrm{Cov}(f(x), g(x))$	Covariance of $f(x)$ and $g(x)$ under $P(\mathrm{x})$
$H(\mathrm{x})$	Shannon entropy of the random variable x
$D_{\mathrm{KL}}(P\\|Q)$	Kullback-Leibler divergence of P and Q
$\mathcal{N}(\boldsymbol{x}; \boldsymbol{\mu}, \boldsymbol{\Sigma})$	Gaussian distribution over $\boldsymbol{x}$ with mean $\boldsymbol{\mu}$ and covariance $\boldsymbol{\Sigma}$

Functions

$f : \mathbb{A} \rightarrow \mathbb{B}$	The function f with domain $\mathbb{A}$ and range $\mathbb{B}$
$f \circ g$	Composition of the functions f and g
$f(\boldsymbol{x}; \boldsymbol{\theta})$	A function of $\boldsymbol{x}$ parametrized by $\boldsymbol{\theta}$. (Sometimes we write $f(\boldsymbol{x})$ and omit the argument $\boldsymbol{\theta}$ to lighten notation)
$\log x$	Natural logarithm of x
$\sigma(x)$	Logistic sigmoid, $\frac{1}{1+\exp(-x)}$
$\zeta(x)$	Softplus, $\log(1+\exp(x))$
$\\|\boldsymbol{x}\\|_p$	L^p norm of $\boldsymbol{x}$
$\\|\boldsymbol{x}\\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\mathrm{condition}}$	is 1 if the condition is true, 0 otherwise
$\arg\max_{\mathbf{x}\in\mathbb{R}^n} f$	
$\arg\min_{\mathbf{x}\in\mathbb{R}^n} f$	
$\lambda, \mathrm{rectifier}, \mathrm{softmax}, \sigma, \zeta, \mathrm{Var}, \mathrm{SE}$	