A Comprehensive Review and Analysis of Deep Learning Techniques for Solving Partial Differential Equations: Performance and Limitations in Higher Dimensions

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

Keren Li *keren.li-2@student.manchester.ac.uk*
Department of Computer Science
The University of Manchester

Contents

List of Figures

List of Tables

Abstract

In recent years, neural networks (NNs) have been increasingly utilized to parametrize the solution of partial differential equations (PDEs). This report present a detailed review and summary of deep learning techniques for solving various types of partial differential equations and their applications in various engineering sciences, and provide a detailed analysis and comparison of the advantages and disadvantages of each method. Physics-Informed Neural Networks (PINNs) have gained prominence, Deep Galerkin Method (DGM), a special case of PINNs based on residual minimization, is compared with energy minimization methods such as the Deep Ritz Method (DRM). However, multiple loss function types can be proposed for the same PDE, and determining the most effective approach remains a challenge. This report investigates the performance of various neural network solution methods for the Poisson equation with Dirichlet boundary conditions, focusing on six different settings that combine elements of DRM and DGM. The primary emphasis is on comparing the performance of these settings across different dimensions, with the main approach for evaluating their effectiveness involving a comparison of empirical loss and fractional error. In the optimal combination, the fractional error attains a remarkable result of 10^{-4} in ten dimensions. Additionally, this report explores the performance and limitations of these six settings at higher dimensions, providing valuable insights into their potential application to more complex problems.
Keywords: Deep Ritz Method; Physics-Informed Neural Networks; High-dimensional.

Acknowledgements

I would like to express my deepest gratitude to my supervisor, Dr. Anirbit Mukherjee, for his invaluable guidance, encouragement, and support throughout the course of this project. As an expert in the mathematics of neural networks and deep learning, his expertise, insight, and constructive feedback have been instrumental in shaping my understanding of this project.

I am truly grateful for the time and effort he has invested in helping me develop my ideas, refine my methodology, and overcome the challenges I encountered along the way. His unwavering belief in my abilities and commitment to my growth as a researcher have been a constant source of motivation and inspiration.

I would also like to extend my thanks to the entire Computer Science Department for providing a stimulating and nurturing academic environment, as well as my fellow colleagues and friends for their camaraderie and support.

Lastly, I am grateful to my family for their love, understanding, and unwavering belief in my potential. Their encouragement has been a driving force behind my pursuit of academic excellence, and I dedicate this work to them.

1 Introduction

1.1 Background

1.1.1 Partial differential equations

Differential equations play a fundamental role in describing the behavior of natural, social, and economic phenomena over time. For instance, the erosion of mountain ranges, riverbed evolution, and population migration can all be modeled using differential equations that involve time as an independent variable. Differential equations relate the behavior of an unknown function to its derivative and the independent variable, and they come in various forms such as ordinary differential equations (ODEs), partial differential equations (PDEs), stochastic differential equations (SDEs), integral differential equations (IDEs), and differential algebraic equations (DAEs).

PDEs are a more general form of differential equations that can describe a wider range of physical laws, making them more complex to solve and of greater research value. Some widely studied PDEs include Maxwell's equation, Schrödinger's equation, Navier-Stokes equation and Poisson's equation in physics, as well as the Hamilton-Jacobi-Bellman equation for engineering applications. Thus, differential equations are essential for modeling physical phenomena in various fields, including natural science, livelihood, economy, and industry. The study of their solution methods has significant theoretical and practical engineering applications.

During the mid-18th century, significant contributions to the study of partial differential equations were made by Swiss mathematician Daniel Bernoulli. Bernoulli developed a general method for understanding the vibration of elastic systems and introduced the concept of "separation of variables." This technique is still widely used in solving partial differential equations. Additionally, French mathematician Joseph Louis Lagrange enriched the field by introducing the theory of characteristics, allowing for the solution of first-order partial differential equations. In the 19th century, mathematicians Augustin-Louis Cauchy and Karl Weierstrass also made important advances when examining partial differential equations, laying the groundwork for many of the techniques used today.

For modelling a wide range of physical and natural events, partial differential equations are essential tools. By solving these equations, researchers can gain insight into the behavior of the system being modeled. Traditionally, analytical methods have been used to solve differential equations, which often require a deep understanding of calculus and mathematical techniques.

1.1.2 Deep learning approaches for solving partial differential equations

Obtaining analytical solutions to differential equations can be a challenging task, and in many cases, such solutions may not exist. However, there are numerous numerical techniques available for solving differential equations. These methods involve approximating the solution to the differential equation using iterative algorithms that generate a sequence of increasingly accurate approximations.

For several decades, data-driven methods have been used as an alternative to traditional methods for solving differential equations. In the 1950s and 1960s, early developments focused on finite difference schemes for partial differential equations (LeVeque, 2007). More recently, in the 1990s, neural networks were introduced as a powerful tool for solving PDEs. This method involves training neural networks on known solutions, a technique known as neural network-based regression (Lagaris et al., 1998). It has proven successful in solving various PDEs, including those related to fluid dynamics, heat transfer, and structural mechanics.

Another data-driven approach for solving differential equations is Gaussian process regression (Calderhead et al., 2008). This technique involves modeling the solution of the differential equation as a Gaussian process and using training data to estimate the process's parameters. Gaussian process regression has shown promise in solving problems related to fluid flow control and inverse heat conduction.

Modeling and predicting complex dynamical systems described by differential equations remain challenging problems. Taking the Earth system as an example, its dynamical properties are influenced by the interaction of physical, chemical, and biological processes occurring at spatial and temporal scales spanning multiple orders of magnitude (Hart & Martinez, 2006). Over the past few decades, researchers have attempted to solve PDEs numerically using various methods, including the finite difference method (Thomas, 2013), the

finite element method (Donea & Huerta, 2003), the spectral method (Feit et al., 1982), and the mesh-free method (Liu & Gu, 2005). These PDEs have been successfully applied to many dynamical systems. However, traditional methods have limitations in terms of their applicability to nonlinear systems of high dimensionality. Additionally, solving the inverse differential equation problem is still a challenging task. In practical applications, data often have missing, cracked, or noisy boundary conditions, significantly affecting the accuracy of traditional methods. Therefore, new methods must be proposed to address these problems.

In recent years, artificial intelligence (AI) and related intelligent technologies have witnessed significant development, with deep learning emerging as a prominent branch of AI (LeCun et al., 2015). Deep learning has shown impressive performance in handling high-dimensional and complex structured data, making it an appropriate tool for dealing with complex system problems. It has excelled in traditional AI tasks such as computer vision, natural language processing, and speech recognition. Furthermore, it has made notable contributions to fields such as Go games (Silver et al., 2017), protein structure prediction (Jumper et al., 2021), particle accelerator data analysis (Ciodaro et al., 2012), and brain circuit reconstruction (Helmstaedter et al., 2013). Particularly, large-scale deep learning models have demonstrated exceptional performance in natural language and image tasks such as topic classification, sentiment analysis, question answering, language translation, and image generation, and are considered a promising tool in achieving general artificial intelligence (Bommasani et al., 2021).

In general, both traditional machine learning and deep learning aim to use data to understand complex systems and achieve specific goals. The primary difference lies in the feature engineering process. Traditional machine learning requires manual feature engineering, where experts must carefully design features that will be used by machine learning algorithms, such as neural networks, support vector machines (SVMs), and decision trees, to accomplish tasks. In contrast, deep learning integrates the feature engineering process into neural networks through the use of complex architectures such as convolutional neural networks (CNNs), recurrent neural networks (RNNs), Transformers, and other advanced models. This approach enables end-to-end learning, where the neural network learns how to extract relevant features from the raw data, allowing for more effective exploitation of the full potential of the neural network and significantly improving its performance (LeCun et al., 2015).

Despite the significant progress made in deep learning, it cannot be ignored that the current success of this approach heavily relies on high-quality training datasets. Datasets such as PASCAL VOC, COCO, ParallelEye-CS, EnglishWikipedia Corpus, among others, have been instrumental in advancing the field of deep learning (Li et al., 2019b; Everingham et al., 2010). Nevertheless, as the complexity of systems continues to increase, and the volume of available data explodes, purely data-driven deep learning models are not delivering the expected performance gains. Even when such models fit results well on a particular dataset, factors such as data noise or observation bias can lead to poor model generalization performance. Moreover, some models may not be practical in real-world applications. To address these issues, additional expert knowledge needs to be incorporated outside of the data-driven framework. This involves introducing "a priori/expert knowledge" to deep learning models, such as reliable theoretical constraints and inductive bias based on observed data. By incorporating this additional knowledge, deep learning models can be guided to provide more accurate and generalizable results (Karniadakis et al., 2021). In the natural sciences, mathematical and physical laws can be considered as expert knowledge, and incorporating such laws or constraints into machine learning models is known as physics-informed machine learning. This approach involves using both physical laws and data to guide the learning process, and the typical network used is the physics-informed neural network (PINN) (Raissi et al., 2019). Researchers have been experimenting with neural network-based or related machine learning methods to solve differential equations since the 20th century, achieving good results (Lagaris et al., 1998). However, there was no uniform name for this approach until PINN was proposed by Raissi et al. (2019), which has gradually become the standard term for this approach. Physically informed machine learning begins with the construction of interpretable neural networks that incorporate relevant physical laws or constraints. This is particularly important when working with imperfect data, such as small amounts of data or data that contains noise or observation bias. By incorporating physical constraints into the model, it can be validated and generalized, even in the presence of imperfect data. Moreover, the predictions generated by the model are consistent with objective physical constraints. In recent years, a cross-cutting area called "AI for Science" has emerged as a new field aimed at

leveraging artificial intelligence (AI) to accelerate research in fundamental disciplines such as mathematics, physics, and chemistry. This area has gained significant attention and has been the focus of workshops at major AI conferences such as the Conference on Neural Information Processing Systems (NeurIPS) and the International Conference on Machine Learning (ICML). These workshops aim to facilitate collaboration and discussion between researchers from both the AI and scientific communities.

1.1.3 Recent advances in deep learning for solving PDEs: domain decomposition techniques

In a number of engineering disciplines, including computer vision and natural language processing, deep learning has produced outstanding results. However, many scientists are currently focusing on the problem of solving partial differential equations using neural networks as models. Several approaches have been developed from different perspectives, including the Deep Galerkin method (Sirignano & Spiliopoulos, 2018), Deep Ritz method (Weinan & Bing, 2017), PDE-Net (Long et al., 2018; 2019), and physically informed neural networks (Raissi et al., 2019). Early work and books on the subject can be found in Lagaris et al. (1998); Yadav et al. (2015), and there is also a significant amount of research on the application of deep learning to specific equation problems (Raissi et al., 2020; Beck et al., 2020).

Physically informed neural networks aim to find the parameters of the neural network by embedding equation constraints into the architecture of the neural network. They utilize the starting point and boundary conditions as the loss function's penalty terms and minimize the loss function using optimization methods such as gradient descent (Raissi et al., 2019). Due to the straightforward and algorithmically effective idea of neural networks based on physical information, they can be applied to problems such as stochastic differential equations (Nabian & Meidani, 2019; Zhang et al., 2019; Yang & Perdikaris, 2019), fractional order differential equations (Pang et al., 2019) and more. Using neural network models, solve partial differential equations numerically are meshless and have obvious advantages in facing problems with high and complex regions (Berg & Nyström, 2018; Zhu et al., 2019).

However, since the neural network model assumes the continuity of functions, it cannot directly deal with the interface conditions of jumps in heterogeneous partial differential equations. As a result, it is unable to finely approximate the discontinuous solution.

The referenced work aims to tackle the challenge of solving heterogeneous partial differential equations defined over complex domains by integrating deep learning and domain decomposition techniques. There are three primary motivations behind this approach. First, classical domain decomposition algorithms relying on grid discretization are difficult to apply to equations over intricate domains, whereas neural networks, as gridless methods, can be employed for getting partial differential equation's result. Second, the implementation complexity of domain decomposition algorithms grows with the increase in the number of subdomains and/or layers, whereas neural network programs are reusable. Finally, the use of deep learning to solve heterogeneous PDEs is an emerging field that warrants further exploration and research. Drawing from these motivations, a novel algorithmic framework, known as the deep domain decomposition method, is introduced. This approach is expected to inherit the advantages of both domain decomposition techniques and deep learning, and it can be extended to address complex coupled model problems. Related works that merge the concepts of deep learning and domain decomposition methods can be found in Li et al. (2019a; 2020); Jagtap & Karniadakis (2021); Heinlein et al. (2021).

1.2 Deep learning approaches for solving complex differential equations

In recent years, researchers from around the world have focused on utilizing deep learning methods to address complex differential equations that emerge in engineering applications, achieving significant results. This section highlights several studies that have successfully employed neural networks to solve differential equations in real-world scenarios.

In the industrial field, numerous problems can be represented by differential equations. Osorio et al. (2022) conducted a study on the application of deep operator networks in photovoltaic power systems. They used historical data of predicted values to replace the temporal coordinates in the backbone network and proposed a data-driven machine learning modeling framework. A case study demonstrated that their method could achieve high-accuracy predictions of the state of photovoltaic power systems at a lower computational cost. Franklin et al. (2022) employed physically informed neural networks to construct virtual sensors as an

alternative to conventional sensors in oil well systems. These virtual sensors were capable of incorporating prior knowledge related to system dynamics to train long short-term memory (LSTM) neural networks without the need to measure all states.

bin Shi & Meng (2022) introduced an end-to-end deep learning framework that simultaneously learns Koopman embedded functions and Koopman operators to address the design challenges of Koopman functions for nonlinear systems. Goswami et al. (2019) investigated the brittle fracture prediction problem based on the phase field method. Unlike other PINN algorithms that minimize the residuals of the control equations, they used the variational energy of the system as the loss function. They modified the neural network output to eliminate the boundary condition loss component in the loss function, enabling the network to fully satisfy the boundary conditions. Zhang et al. (2020) employed deep neural networks to try to approximate the forward problem's solution and the material parameters that are unknown, respectively. They constructed physically informed deep neural networks using deep learning methods to solve the identification problem of non-uniform materials.

In the field of economics, Kolmogorov partial differential equations and stochastic differential equations effectively describe the dynamics of economic systems and are extensively utilized in financial system modeling. However, most approximation methods for Kolmogorov differential equations either encounter the issue of dimensional catastrophe or only achieve the approximation of partial differential equations at a single fixed spatial and temporal point.

To address these challenges, Beck et al. (2018) transformed the Kolmogorov partial differential equation solution problem into an infinite-dimensional stochastic optimization problem. They suggested a deep learning approach for resolving Kolmogorov partial differential equations after approximating the equations using a fully connected deep neural network. The feasibility of solving such differential equations through deep learning methods was verified through numerical simulations.

Glau & Wunderlich (2022) investigated the application of deep learning to solve PDE methods in option pricing. They employed an unsupervised training approach, eliminating the need to sample the solution space. The proposed deep parametric PDE method is divided into two phases: offline and online. The offline phase involves training the neural network, while the online phase evaluates the state, parameter values, and sensitivity.

In other scientific fields, the solution of complex differential equations has likewise yielded a large number of results. Barmparis & Tsironis (2020) used physics-informed neural network to analyze the spread of the new crown epidemic in 2020 and made a short-term prediction in weeks, which provided a scientific analysis basis for the epidemic prevention and control.

1.3 Motivation

1.3.1 Project motivation

The growing demand for precise and effective computing methods to solve partial differential equations (PDEs) is what inspired this effort. PDEs play a crucial role in modeling various physical, biological, and engineering processes. Conventional numerical techniques for solving PDEs, such as finite difference, finite element, and finite volume methods, often face limitations in handling complex geometries, nonlinearities, and high-dimensional problems.

With the rapid development of deep learning and neural networks, there is a growing interest in leveraging these powerful tools to address the challenges associated with solving PDEs. Neural networks have demonstrated remarkable success in various fields, including natural language processing and reinforcement learning. As such, adapting these techniques for PDEs has the potential to overcome the limitations of traditional methods and revolutionize the field.

Thus, the primary motivation behind this project is to explore the capabilities of different neural network-based techniques. Solving partial differential equations (PDEs) with variational energy loss, such as the Deep Ritz Method (DRM) (Weinan & Bing, 2017), is an interesting edge case. Two common setups have emerged for addressing PDEs with DRM: one approach involves adding a penalty term to the loss function to enforce boundary conditions (Weinan & Bing, 2017), while the other satisfies boundary conditions directly

within the model (Wu et al., 2021). The latter method circumvents the challenges associated with boundary sampling in high-dimensional problems, but implementing it in high dimensions can be a difficult task in practice.

Deep Ritz Method (DRM) (Weinan & Bing, 2017), and Physics-Informed Neural Networks (PINNs) (Raissi et al., 2019) are two prominent techniques that utilize a single neural network to solve partial differential equations (PDEs), where the Deep Galerkin Method (DGM) (Sirignano & Spiliopoulos, 2018) is a PINNs special case. Upon examining these approaches, it becomes apparent that there are numerous potential combinations of their respective methods. However, determining an optimal way to combine these ideas for specific cases remains a challenge.

The primary goal of this project is to investigate the performance of these methods and determine the most effective approach for solving the Poisson PDE, a common case where both variational energy methods and residual methods can be applied. Notably, the Poisson PDE is chosen as a test case due to its compatibility with various solution techniques, as not all PDEs permit the use of variational formulation. Furthermore, the Poisson equation is inherently defined in every dimension, rendering it an excellent test case for addressing multidimensional problems.

1.3.2 Further motivation of poisson equation

Solving the Poisson PDE is an essential intermediate step in approximating the solution of the Navier-Stokes equations due to its role in enforcing incompressibility and ensuring physically consistent fluid behavior. 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 (Schneiderbauer & Krieger, 2013). These equations, fundamental to comprehending fluid dynamics, are based on the ideas of mass, momentum, and energy conservation. Considering that the Burgers' equation (Mittal & Singhal, 1993) can be considered a simplified version of the Navier-Stokes equations, the Poisson equation's importance extends to the Burgers' PDE as well.

The constant-property, incompressible Navier-Stokes equations are given by (Wang, 1991):

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho}\nabla p + \nu \nabla^2 \mathbf{u} + \mathbf{f}, \tag{1}$$

$$\nabla \cdot \mathbf{u} = 0, \tag{2}$$

where $\mathbf{u}$ is the fluid velocity field, t is time, p is pressure, ρ is fluid density, ν is the kinematic viscosity, and $\mathbf{f}$ represents external body forces.

The first equation represents the momentum equation, while the second equation enforces the incompressibility constraint (continuity equation). In order to satisfy the incompressibility constraint, the pressure field must be determined. This is achieved by taking the divergence of the momentum equation and rearranging terms to obtain the Poisson equation for the pressure field:

$$\nabla^2 p = \nabla \cdot (\rho(\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u})). \tag{3}$$

Solving this Poisson equation for the pressure field ensures that the continuity equation is satisfied, and the resulting velocity field is divergence-free (incompressible). Once the pressure field is known, it can be incorporated back into the momentum equation to update the velocity field (Pletcher et al., 2012).

1.4 Specific test case

The Poisson equation is a fundamental partial differential equation that arises in many areas of physics and engineering. It bears the name of the French mathematician and physicist, Siméon-Denis Poisson, who introduced the equation in the beginning of 19th century. The Poisson equation takes the following form:

$$\nabla^2 u = f(\mathbf{x}), \tag{4}$$

When u denotes the function that needs to be solved, ∇^2 is the Laplacian operator, and $f(\mathbf{x})$ is a given function. The Laplacian operator is defined as:

$$\nabla^2 u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}, \tag{5}$$

where the spatial variables x, y, and z are functions that make up u. The Poisson equation arises in many different physical contexts, such as electrostatics, heat transfer, and fluid mechanics. In electrostatics, for example, the Poisson equation is used to determine the electric potential in a region of space given the charge distribution in that region.

1.4.1 Ritz method and Galerkin method

Now over a bounded domain $\Omega \subset \mathbb{R}^d$, think about the following Poisson equation with Dirichlet boundary condition.

$$\begin{cases} -\Delta u = f, & \text{in } \Omega, \\ u = 0, & \text{on } \partial\Omega. \end{cases} \tag{6}$$

The weak form, also known as the virtual work principle, is given by :

$$\begin{cases} \text{Find} \quad u \in H_0^1(\Omega), \quad s.t. \\ a(u,v) = (f,v), \forall v \in H_0^1(\Omega), \end{cases} \tag{7}$$

where $H_0^1(\Omega)$ denotes the set of admissible functions, $a(u,v)$ and (f,v) are bilinear and linear forms, respectively, defined as $a(u,v) = \int_\Omega \nabla u \cdot \nabla v dx$ and $(f,v) = \int_\Omega f v dx$. Alternatively, the weak form can be expressed as the minimum potential energy principle, is given by:

$$\begin{cases} \text{Find} \quad u \in H_0^1(\Omega), \quad s.t. \\ I(u) = \min\limits_{v \in H_0^1(\Omega)} I(v), \end{cases} \tag{8}$$

where $I(v) = \frac{1}{2}a(v,v) - (f,v)$ is the action functional. To obtain a discrete solution, the trial and test function spaces are replaced by appropriate finite dimensional subspaces, say $V_h(0) \in H_0^1(\Omega)$, which leads to the discrete problem given by:

$$\begin{cases} \text{Find} \quad u_h \in V_h(0), \quad s.t. \\ a(u_h, v_h) = (f, v_h), \forall v_h \in V_h(0). \end{cases} \tag{9}$$

Such an approach is called the Galerkin method. Replace the admissible function space by an appropriate finite dimensional subspaces, say $V_h(0) \in H_0^1(\Omega)$, then led to the discrete problem:

$$\begin{cases} \text{Find} \quad u_h \in V_h(0), \quad s.t. \\ I(u_h) = \min\limits_{v_h \in V_h(0)} I(v_h). \end{cases} \tag{10}$$

Such an approach is called the Ritz method. The Ritz method is a classical approach for solving PDEs, first developed by Walther Ritz in 1909, which involves using a trial solution which satisfies the boundary conditions and minimizing the residual error of the PDE with respect to this trial solution. However, the "curse of dimensionality" limits the effectiveness of the Ritz method for high-dimensional problems (Bellman, 2010). The Deep Ritz method (DRM) addresses this by using neural networks to high-dimensional nonlinear PDEs and approximating their solutions with reasonable computational costs (Weinan & Bing, 2017). The Deep Galerkin method is another deep learning approach that involves approximating the solution using a neural network that satisfies the differential equation instead of boundary conditions at a collection of collocation sites. (Sirignano & Spiliopoulos, 2018). The network weights are then optimized using the residual error of the PDE at these collocation points. Both of these methods have shown promising results in solving nonlinear PDEs in high dimensions, overcoming the curse of dimensionality limitation.

1.5 Summary of the results

Upon comparing these techniques, it is found that infact none of the variational energy method or residual method outperforms the others on their own. Consequently, the research introduces a novel modification to the PINNs approach, combining a multiplicative regularizer to enhance performance. This modification method employs a residual loss without penalties and integrates the penalty term directly into the model.

The method trains a transformed neural network, incorporating an additional scaling factor, instead of training the neural network directly. The study explores three different settings for the variational energy loss function and the model: satisfying the boundary conditions in the model with trivial scaling, satisfying the boundary conditions in the model with non-trivial scaling, and satisfying the boundary conditions in the loss function by penalty term. These three settings are also applied to residual minimization.

Ultimately, the research compares and identifies the best-performing combination, which is the residual minimization with boundary conditions included in the model and an additional scaling factor. This approach not only offers improved performance compared to variational energy method and residual method individually, but also provides a promising direction for future research in solving PDEs using deep learning techniques.

1.6 Structure of the project report

The project report is organized into the following sections:

1. **Introduction**: This section provides a comprehensive background on partial differential equations, deep learning approaches for solving PDEs, and recent advances in domain decomposition techniques. The motivation for the project, focusing on the Poisson equation, is described along with a specific test case involving the Ritz and Galerkin methods. A summary of the results and the structure of the project report are also included.

2. **Theoretical Basis**: This section covers different minimization approaches for solving PDEs, such as energy minimization and residual minimization. The neural network architecture and the Stochastic Gradient Descent Algorithm are also discussed.

3. **Experimental Setup and Parameter Setting**: This section presents the variational energy loss and residual loss setups, including boundary included models with trivial and non-trivial scaling, and models with boundary penalty. The parameters settings for these loss functions are described in detail.

4. **Experiment Results**: In this section, a comprehensive comparison of a 10-dimensional problem is presented, followed by a comparative analysis of dimension-dependent performance with different settings. The conclusions for the dimension 10 problem and the overall performance are discussed.

5. **Summary and Outlook**: The final section provides a summary of the project work and explores possible future developments in the field of deep learning for solving PDEs.

2 Theoretical Basis related to Neural Networks for Solving Partial Differential Equations

2.1 Different minimization approach for solving partial differential equations

2.1.1 Energy minimization

Energy minimization is a classical method for solving PDEs, and it is based on the principle of minimizing the energy functional of the PDE. The energy functional is a mathematical expression that describes the system's overall energy modeled by the PDE. In the context of PDEs, the energy functional is typically defined as an integral over the domain of the PDE, with the integrand consisting of terms involving the unknown function and its derivatives.

The energy minimization method involves finding the function that minimizes the energy functional of the PDE subject to certain boundary conditions. This is usually done by applying the calculus of variations, which is a branch of mathematics that deals with optimization problems involving functionals. The calculus of variations allows one to find the function that minimizes the energy functional by setting its variation to zero.

The Deep Ritz Method (DRM) is an energy minimization method that utilizes deep neural networks to roughly approximate the solution to a Partial Differential Equation (PDE). The method is based on the variational problem stated in Evans (1998):

$$\min_{u \in H} \mathcal{R}(u), \tag{11}$$

where H denotes the set of admissible functions, which is also referred to as the trial function, and u is a function in H. The goal is to reduce the population risk indicated by $\mathcal{R}(u)$:

$$\mathcal{R}_{\text{energy}}(u) = \int_{\Omega} \left(\frac{1}{2} |\nabla u(x)|^2 - f(x)u(x) \right) dx, \tag{12}$$

where the function f represents an external force acting on the system under study. This kind of issue arises frequently in the physical sciences. Moreover, it is worth noting that the lower bound property of the functional $\mathcal{R}_{energy}(u)$ plays a crucial role in the Deep Ritz Method, ensuring the existence of a minimizer. Thus, a brief proof process is as follows. Firstly, the Cauchy-Schwarz inequality is applied to the second term of the functional:

$$\left| \int_{\Omega} f(x)u(x)dx \right| \leq |f|_{L^2(\Omega)} |u|_{L^2(\Omega)}. \tag{13}$$

Then to establish a relationship between the L^2 norm of u and its gradient, Poincaré's inequality is utilized:

$$|u|_{L^2(\Omega)} \leq C |\nabla u|_{L^2(\Omega)}, \tag{14}$$

where C is a positive constant that depends on the domain Ω. Finally, by combining the inequalities from the previous subsections, a lower bound for $\mathcal{R}_{energy}(u)$ is derived:

$$\mathcal{R}_{\text{energy}}(u) \geq \frac{1}{2} |\nabla u|^2_{L^2(\Omega)} - C |f|_{L^2(\Omega)} |\nabla u|_{L^2(\Omega)}. \tag{15}$$

2.1.2 Residual minimization

Residual minimization is a more recent approach to solving PDEs, and it is based on the principle of minimizing the residual of the PDE. The residual is defined as the difference between the actual solution of the PDE and its numerical approximation. In other words, it represents the error in the numerical solution.

The residual minimization method involves finding the function that minimizes the residual of the PDE subject to certain boundary conditions. This is usually done by using an iterative algorithm that updates the numerical approximation of the solution at each step. The algorithm works by computing the residual of the PDE at the current approximation, and then adjusting the approximation to reduce the residual.

The Deep Galerkin Methods (DGM) is an example of a residual minimization method that uses deep neural networks to approximate the solution to a PDE.

$$\mathcal{R}_{\text{residual}}(u) = \int_{\Omega} \left(-\Delta u(x) - f(x) \right)^2 dx. \tag{16}$$

2.2 Neural network

The neural network in this experiment is constructed as (17), where contains one input, four weight layers, three nonlinear transformation operations (activation functions), and one output. The shortcut connection skips one or more layers and directly connects the input to the output of the block. This allows information to bypass the weight layers, which is designed to help prevent the vanishing gradient problem and allow the network to be trained more effectively.

In the experiment, the neural network architecture is used to build the neural network $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}$, where A_1, A_2, A_3, and A_4 are fully-connected layers with the activation function σ. The number of parameters in $\mathcal{N}$ is given by $p = (d \cdot \text{width} + \text{width}) + 2(\text{width} \cdot \text{width} + \text{width}) + (\text{width} \cdot 1 + 1) = 2 \cdot \text{width}^2 + (4 + d) \cdot \text{width} + 1$.

The number of layers and the width of each layer in this architecture can be adjusted depending on the complexity of the PDE and the size of the domain. The experiment define the neural network $\mathcal{N} : \mathbb{R}^d \to \mathbb{R}$ as follows:

$$\mathcal{N}(\boldsymbol{x}) := A_4 \cdot \sigma(A_3 \cdot \sigma(A_2 \cdot \sigma(A_1))), \tag{17}$$

The complete network is denoted by:

$$z_\theta(\boldsymbol{x}) = f_4 \circ \cdots \circ f_1(\boldsymbol{x}), \tag{18}$$

where θ refers to the collection of all the network's parameters. Note that the first block's input x is in $\mathbb{R}^d$.

Having z_θ, the function u can be obtained by

$$u(\boldsymbol{x}; \theta) = a \cdot z_\theta(\boldsymbol{x}) + b, \tag{19}$$

where in the left-hand side and in the following discussion, θ is used to refers to the entired parameter set $\{\theta, a, b\}$. The function of θ that results from substituting this into the form of $\mathcal{R}$ should be minimized.

Regarding the energy function that takes place in Equation (12), denote:

$$g(\boldsymbol{x}; \theta) = \frac{1}{2}|\nabla u(\boldsymbol{x}; \theta)|^2 - f(\boldsymbol{x})u(\boldsymbol{x}; \theta). \tag{20}$$

The optimization problem can then be formulated as:

$$\min_\theta \hat{\mathcal{R}}_{\text{energy}}(\theta), \quad \hat{\mathcal{R}}_{\text{energy}}(\theta) = \int_\Omega g(\boldsymbol{x}; \theta) dx. \tag{21}$$

To implement residual minimization using the neural network, regarding the residual function that takes place in Equation (16), denote:

$$r(\boldsymbol{x}; \theta) = -\Delta u(\boldsymbol{x}; \theta) - f(\boldsymbol{x}), \tag{22}$$

the optimization problem can be formulated as follows:

$$\min_\theta \hat{\mathcal{R}}_{\text{residual}}(\theta), \quad \hat{\mathcal{R}}_{\text{residual}}(\theta) = \int_\Omega (r(\boldsymbol{x}; \theta))^2 dx, \tag{23}$$

In both cases, the approximated solution $u(\boldsymbol{x}; \theta)$ is represented as the output of the neural network with parameters θ. The neural network takes the input $\boldsymbol{x} \in \Omega$ and produces an output $u(\boldsymbol{x}; \theta)$. The Laplacian of $u(\boldsymbol{x}; \theta)$ with respect to $\boldsymbol{x}$ is computed using automatic differentiation techniques.

By minimizing the residual functional, the neural network is trained to find an approximation that satisfies the PDE as closely as possible. The neural network architecture, including the number of layers and neurons per layer, can be adjusted to improve the accuracy and performance of the residual method.

2.3 Stochastic Gradient Descent Algorithm

In the context of neural network training, Stochastic Gradient Descent (SGD) is a widely employed optimization method. For smaller data sizes, these techniques facilitate gradient calculation on the entire dataset and allow for iterative updates. However, when the data size increases, calculating gradients for the complete dataset in each iteration becomes computationally expensive. Owing to its capacity to significantly reduce computational overhead, SGD is frequently utilized for optimization in large-scale data scenarios.

In the experiment, the SGD algorithm for the variational energy minimization is given by:

$$\theta^{k+1} = \theta^k - \eta \nabla_\theta \frac{1}{N} \sum_{j=1}^{N} g(\boldsymbol{x}_{j,k}; \theta^k), \tag{24}$$

In the case of residual minimization, using the SGD algorithm for residual minimization, the update equation becomes:

$$\theta^{k+1} = \theta^k - \eta \nabla_\theta \frac{1}{N} \sum_{j=1}^{N} r(\boldsymbol{x}_{j,k}; \theta^k)^2, \tag{25}$$

where, for each k, $\boldsymbol{x}_{j,k}$ is selected randomly from a uniformly spaced grid of points. The method employs a Stochastic Gradient Descent technique, utilizing a mini-batch and an Adam optimizer (Kingma & Ba, 2014), which is prevalent in deep learning tasks and consistently applied in the experiments presented throughout this report.

The main difference between the two SGD algorithms lies in the objective function being minimized: for variational energy minimization, it is $g(\boldsymbol{x}; \theta)$, while for residual minimization, it is $r(\boldsymbol{x}; \theta)^2$.

3 Experimental Setup and Parameter Setting

Consider the Poisson equation on $\Omega = (0, 1)^d$:

$$\begin{cases} -\Delta u = f(\boldsymbol{x}), & \text{in } \Omega, \\ u(\boldsymbol{x}) = 0, & \text{on } \partial\Omega. \end{cases} \tag{26}$$

The forcing term f is specified by assuming the form of the solution first. The experiment assume the exact solution:

$$u(\boldsymbol{x}) = \prod_{i=1}^{d} \sin(\pi x_i), \tag{27}$$

with $\boldsymbol{x} = (x_1, x_2, \cdots, x_d)$, then $f(\boldsymbol{x}) = d\pi^2 \prod_{i=1}^{d} \sin(\pi x_i)$. $u(\boldsymbol{x})$ that satisfies both the partial differential equation ($-\Delta u = f$) and the boundary conditions (u = g on $\partial\Omega$). The solution $u(\boldsymbol{x})$ represents the distribution of a physical quantity, such as temperature or pressure, that satisfies the equation and the conditions.

Fractional errors are frequently employed in numerical methods for solving differential equations to evaluate the accuracy of the numerical solution (Glusa & Otárola, 2021). These errors provide a normalized measure of the difference between the predicted solution and the exact solution, allowing for comparison of the accuracy of various numerical methods or different parameter settings for a specific method. In the forthcoming experiment, the Fractional-Error will be utilized to assess the numerical error:

$$\text{Fractional–Error} = \frac{\|\hat{u}(\boldsymbol{x}; \theta) - u(\boldsymbol{x})\|_2^2}{\|u(\boldsymbol{x})\|_2^2} \tag{28}$$

Where $\|\cdot\|_2$ denotes the L_2 norm for function of $\boldsymbol{x}$ (Garbled, 2014). $\hat{u}(\boldsymbol{x}; \theta)$ is the Network approximation in the minimization procedure given by variational energy minimization or residual minimization, and u(x) is the exact solution. It is worth noting that in the case $\hat{\mathcal{R}}_{\text{energy,loss–with–boundary–penalty}}(\mathcal{N})$ (34) and

$\hat{\mathcal{R}}_{\text{residual,loss-with-boundary-penalty}}(\mathcal{N})$ (39), $\hat{u}(\boldsymbol{x};\theta)$ replaced by $\mathcal{N}(\boldsymbol{x};\theta)$, where directly use the Network output.

The error calculation calculates the difference between the predicted output of the model and the exact output, and returns the quotient of the numerator and denominator, giving the fractional error. It is useful in evaluating the accuracy of the model and identifying areas of improvement. In order to fully investigate the performance of energy minimization and residual minimization for this specific problem, the experiment conducted in this study employs six different settings for the loss function and model.

3.1 Variational energy loss setups

The first focus of this experiment is on investigating the settings necessary to satisfy boundary conditions in the variational energy minimization method, while keeping other settings constant. The constraints imposed by this approach are subsequently analyzed for high-dimensional problems, highlighting potential limitations when dealing with such complexities. Specifically, an increase in dimensionality leads to a significant increase in fractional error in higher dimensions. In order to improve the performance of the higher dimensions in this setup, the experiment then add a scaling factor which focus on normalize the output of the model with various dimensions. Nevertheless, Weinan & Bing (2017) also suggests that the DRM model should exhibit favorable performance in high-dimensional settings. Thus finally, the same setting as Weinan - adding a penalty term to the loss function - is explored in this performance of this particular PDE problem.

3.1.1 Variational energy loss with boundary included model with trivial scaling

In the first experimental setup, the Dirichlet boundary condition is enforced by directly incorporating it into the model. In this case, the boundary included model is used.

$$\text{model} = \hat{u}(\boldsymbol{x};\theta) = \lambda^d \cdot \Big(\prod_{i=1}^{d}(1-x_i)x_i\Big) \cdot \mathcal{N}(\boldsymbol{x};\theta) = \prod_{i=1}^{d}(1-x_i)x_i \cdot \mathcal{N}(\boldsymbol{x};\theta), \tag{29}$$

where $\mathcal{N}(\boldsymbol{x};\theta)$ is a function represented by a neural network. In the first term, a scaling factor λ is introduced to normalize the model output, and at this initial setup $\lambda = 1$, which results in an un-normalized output. Trivial scaling is used later in this experiment which refer to this setting. The second term is a simple polynomial function that ensures the model satisfies the boundary conditions $u(\boldsymbol{x}) = 0$ when $x_i = 0$ or $x_i = 1$ for any $i = 1, \ldots, d$. The neural network function $\mathcal{N}(\boldsymbol{x};\theta)$ is parameterized by θ, which represents the weights and biases of the neural network. This results in the calculation of the variational energy loss, denoted by $\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=1}(\hat{u})$, which is given by

$$\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=1}(\hat{u}) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{2}\left|\nabla\hat{u}(\boldsymbol{x}_i;\theta)\right|^2 - f(\boldsymbol{x}_i)\hat{u}(\boldsymbol{x}_i;\theta)\right], \quad \text{where } \boldsymbol{x}_i \in [0,1]^d, \tag{30}$$

where N represents the total number of mesh points used in the training dataset, and i is the index used to iterate over the mesh points.

3.1.2 Variational energy loss with boundary included model with non-trivial scaling

In the second experimental setup, the same conditions are followed to ensure that the boundary conditions are satisfied within the model. During training, it was observed that at the initial setup, the output values of the model were significantly reduced in high-dimensional settings, resulting in a persistent fractional error of one. To improve the model's performance in such settings while still adhering to the boundary conditions, the normalization term λ is modify to 5 to the model to ensure that the output of the neural network is scaled correctly. Specifically, the model is defined as

$$\text{model} = \hat{u}(\boldsymbol{x};\theta) = 5^d \cdot \Big(\prod_{i=1}^{d}(1-x_i)x_i\Big) \cdot \mathcal{N}(\boldsymbol{x};\theta), \tag{31}$$

where $\mathcal{N}(x;\theta)$ is a function represented by a neural network. In dimensions 1, 2, 3, and 10, $\lambda = 5$ is selected. Non-trivial scaling/scaling is used later in this experiment which refer to this setting. In this case, the energy loss function in Equation (32) is used.

$$\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=5}(\hat{u}) = \frac{1}{N}\sum_{i=1}^{N}\left[\frac{1}{2}\left|\nabla\hat{u}(\boldsymbol{x}_i;\theta)\right|^2 - f(\boldsymbol{x}_i)\hat{u}(\boldsymbol{x}_i;\theta)\right], \quad \text{where } \boldsymbol{x}_i \in [0,1]^d, \tag{32}$$

3.1.3 Variational energy loss with boundary penalty

In the third experimental setup, the experiment aims to enforce the boundary conditions of the partial differential equation (PDE) by modifying the loss function (Weinan & Bing, 2017). The penalty term added to the loss function encourages the neural network to satisfy the boundary conditions by penalizing the difference between the predicted solution and the known boundary conditions at the domain boundary. The modified functional is defined as follows:

$$\mathcal{R}_{\text{energy,loss-with-boundary-penalty}}(\mathcal{N}) = \int_{\Omega} \left(\frac{1}{2} |\nabla \mathcal{N}(\boldsymbol{x};\theta)|^2 - f(\boldsymbol{x})\mathcal{N}(\boldsymbol{x};\theta) \right) dx + \beta \int_{\partial\Omega} \mathcal{N}(\boldsymbol{x};\theta)^2 ds, \tag{33}$$

To be more precise, the loss function is formulated as:

$$\hat{\mathcal{R}}_{\text{energy,loss-with-boundary-penalty}}(\mathcal{N}) = \frac{1}{N} \sum_{i=1}^{N} \left[\frac{1}{2} |\nabla \mathcal{N}(\boldsymbol{x}_i;\theta)|^2 - f(\boldsymbol{x}_i)\mathcal{N}(\boldsymbol{x}_i;\theta) \right] + \frac{\beta}{M} \sum_{j=1}^{M} \mathcal{N}(\boldsymbol{x}_j;\theta)^2, \ \boldsymbol{x}_i \in [0,1]^d, \tag{34}$$

where N represents the total number of mesh points used in the training data-set, and M is the number of points on the boundary of the domain Ω. In this case, the loss directly use the network output $\mathcal{N}$ instead of model. The index i is used to iterate over the mesh points, while the index j is used to iterate over the boundary points.

3.2 Residual loss setups

In a similar fashion to the energy loss, the residual loss was also evaluated in the three different settings described earlier in the experiment.

3.2.1 Residual loss with boundary included model with trivial scaling

In the first setting, the boundary included model is used, with the initial setup scaling factor $\lambda = 1$,

$$\text{model} = \hat{u}(\boldsymbol{x};\theta) = \lambda^d \cdot \left(\prod_{i=1}^{d} (1 - x_i)x_i\right) \cdot \mathcal{N}(\boldsymbol{x};\theta) = \prod_{i=1}^{d} (1 - x_i)x_i \cdot \mathcal{N}(\boldsymbol{x};\theta), \tag{35}$$

and the residual loss, denoted by $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=1}(\hat{u})$, is given by:

$$\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=1}(\hat{u}) = \frac{1}{N} \sum_{i=1}^{N} \left(-\Delta \hat{u}(\boldsymbol{x}_i;\theta) - f(\boldsymbol{x}_i) \right)^2, \quad \text{where } \boldsymbol{x}_i \in [0,1]^d. \tag{36}$$

The loss function is the mean squared difference between the Laplacian of the predicted function $\hat{u}$ and the source term f at each point x_i, where N is the total number of points in the domain $[0,1]^d$. Similar to Energy before, the boundary included model used in this case is same as Equation (29).

3.2.2 Residual loss with boundary included model with non-trivial scaling

In the second setting, the boundary conditions were again enforced in the model, but with a scaling factor introduced to normalize the output in higher dimensions, as described earlier. In this case, the experiment use the model:

$$\text{model} = \hat{u}(\boldsymbol{x};\theta) = 5^d \cdot \left(\prod_{i=1}^{d} (1 - x_i)x_i\right) \cdot \mathcal{N}(\boldsymbol{x};\theta), \tag{37}$$

and the residual loss function:

$$\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=5}(\hat{u}) = \frac{1}{N} \sum_{i=1}^{N} \left(-\Delta \hat{u}(\boldsymbol{x}_i;\theta) - f(\boldsymbol{x}_i) \right)^2, \quad \text{where } \boldsymbol{x}_i \in [0,1]^d. \tag{38}$$

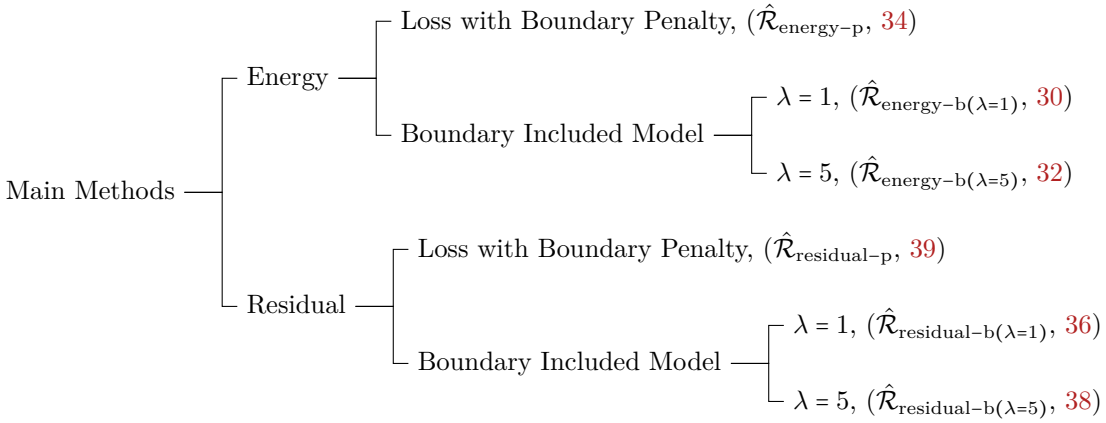
3.2.3 Residual loss with boundary penalty

Similarly, the third setting is a modification of the residual loss approach that uses the penalty term to enforce the boundary conditions of the PDE. This is done by adding a term to the loss function that encourages the network to satisfy the boundary conditions, which also use the network $\mathcal{N}$ directly instead of model. The modified loss function used in this setting is given by:

$$\hat{\mathcal{R}}_{\text{residual,loss-with-boundary-penalty}}(\mathcal{N}) = \frac{1}{N}\sum_{i=1}^{N}\left(-\Delta\mathcal{N}(\boldsymbol{x}_i;\theta) - f(\boldsymbol{x}_i)\right)^2 + \frac{\beta}{M}\sum_{j=1}^{M}\mathcal{N}(\boldsymbol{x}_j;\theta)^2,\ \boldsymbol{x}_i \in [0,1]^d, \quad (39)$$

where $\mathcal{N}(\boldsymbol{x}_i;\theta)$ is the predicted solution at the i-th point in the mesh, N is the total number of mesh points used in the training dataset, $f(\boldsymbol{x}_i)$ is the source term at the i-th point, M is the number of points on the boundary of the domain Ω, the index i is used to iterate over the mesh points, while the index j is used to iterate over the boundary points. β is a hyperparameter that controls the strength of the penalty term.

Overall, the relationship of these six settings is as follows, while a set of streamlined notations will be introduced for utilization at a later stage.



3.3 Parameters Setting

In this section, the experimental results indicate an intriguing observation: the sensitivity of hyperparameters is not significantly affected by the choice between variational energy and residual methods. Instead, the dependency is more pronounced in relation to whether the loss function incorporates boundary penalties or includes boundary conditions within the model. During the experiments, distinct parameter configurations were implemented for each of the setups, which were categorized based on whether the loss function encompasses boundary penalties or integrates boundary conditions within the model. Although the training speed varies between them, the same number of epochs is used, as it is sufficient for the losses to converge in both cases. For all dimension $1, 2, 3$ and 10 problems, observations were made by varying the width in this experiment, which the network width grows from 25 at intervals of 25, up to 150. The experiment also plots the trend of empirical loss and fractional error with increasing epoch to determine the fitting state of the neural network. It is noteworthy that throughout this experiment, a fixed learning rate of 0.0001 has been employed.

3.3.1 Loss with boundary included model

In this experiment, a training period of 2000 epochs was utilized for 1, 2, and 3 dimensions, whereas 10 dimensions underwent 7000 epochs. The training samples were progressively increased based on dimensions, with an initial value of 1000 at dimensions $1, 2, 3$, increasing to 5000 at dimension 10. With the boundary included model, it was observed that the presence or absence of a scale factor did not affect the fitting speed in lower dimension($d = 1, 2, 3$). Results of the fit with trivial scaling are presented below in Figure 1 and 2, where the fractional error shows that at low dimensions such as $d = 3$, both energy and residual losses are fitted at around 400 epochs when the number of parameters is $2w^2 + (d + 4)w + 1 = 46051$. At $d = 10$, the energy loss is fitted at 6000 epochs, and the residual loss reaches a similar level at around 5000 epochs, at which point the total parameter is $2w^2 + (d + 4)w + 1 = 47101$. From the above two graphs, it is obvious

that the higher the dimensionality of the differential equation, the more samples it needs and the more times it needs to be trained to achieve a better fit, which means that there is a dimensional catastrophe in the high-dimensional space. This also reflects the difficulty of fitting high-dimensional spaces, which requires more skill in tuning parameters, as well as a network structure with more parameters and more complexity.

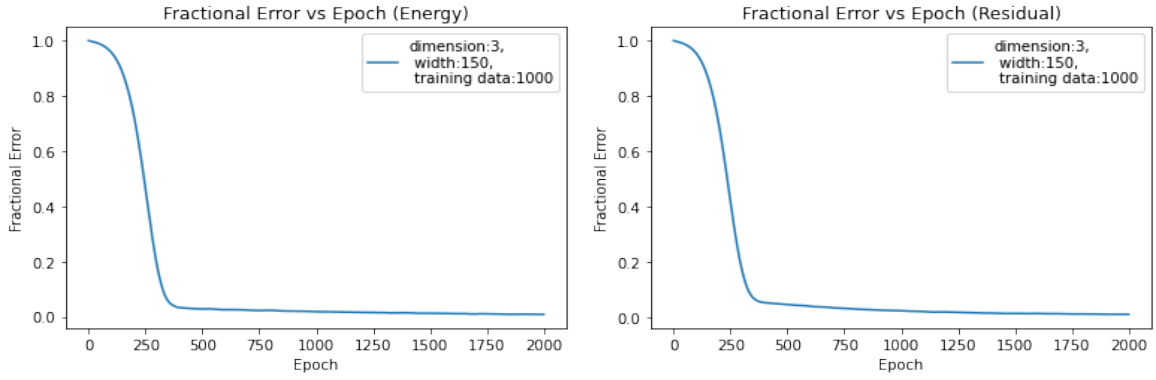


Figure 1: Error with epoch performance ($d = 3$). *Left* : $\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=1}$ (30). *Right*: $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=1}$ (36).

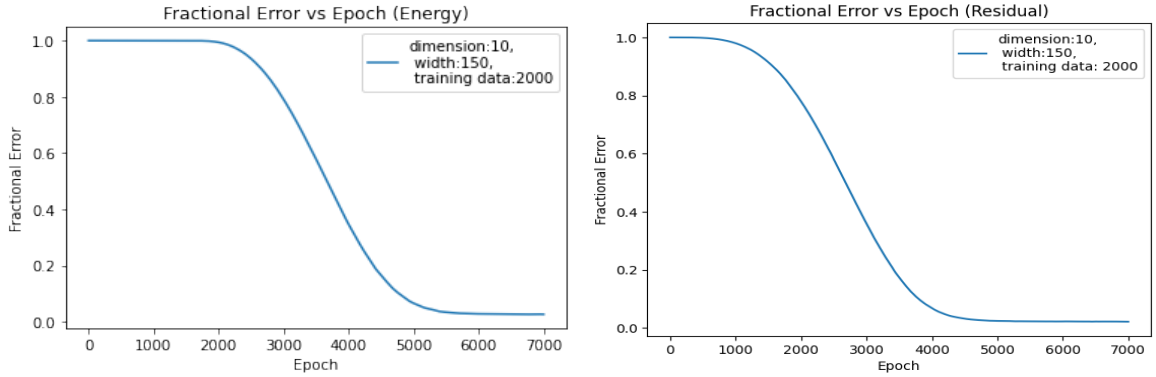


Figure 2: Error with epoch performance ($d = 10$). *Left* : $\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=1}$ (30). *Right*: $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=1}$ (36).

3.3.2 Loss with boundary penalty

In the experimental setting when the loss function was augmented with a boundary penalty, a faster convergence rate was observed in both variational energy and residual loss, with the effect being more pronounced in the case of ten dimensions. Similarly, the training period for 1, 2, and 3 dimensions consisted of 2000 epochs, while 10 dimensions underwent 7000 epochs. The experiment sample 10,000 mesh points in Ω and 200 points at each hyperplane that composes $\partial\Omega$.

The experiment made an important observation regarding the choice of hyperparameters for training the energy and residual loss. Specifically, it was found that selecting a β value of 50 yielded more favorable results when training for the energy loss, whereas a β value of 200 led to better results for the residual loss. In Figure 3 and 4, it can be observed that the energy loss penalty is achieved in approximately 1000 epochs in the case of 10 dimensions, indicating a very rapid convergence. In contrast, the residual loss takes around 4000 epochs to achieve a similar result.

To provide further clarity, Table 1 presents the hyperparameter settings corresponding to the six loss function settings previously mentioned. The reason behind increasing the network widths to 250 is discussed in Section

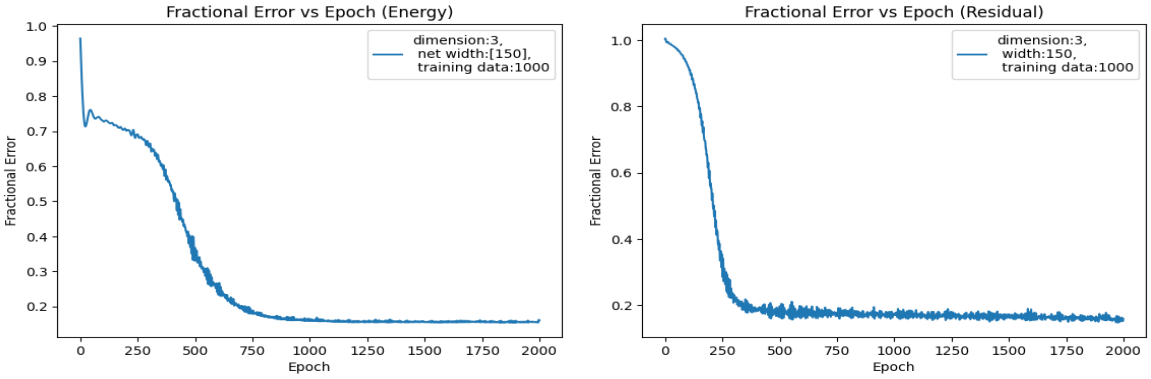


Figure 3: Error with epoch performance ($d = 3$). *Left* : $\hat{\mathcal{R}}_{\text{energy,loss-with-boundary-penalty}}$ (34). *Right*: $\hat{\mathcal{R}}_{\text{residual,loss-with-boundary-penalty}}$ (39).

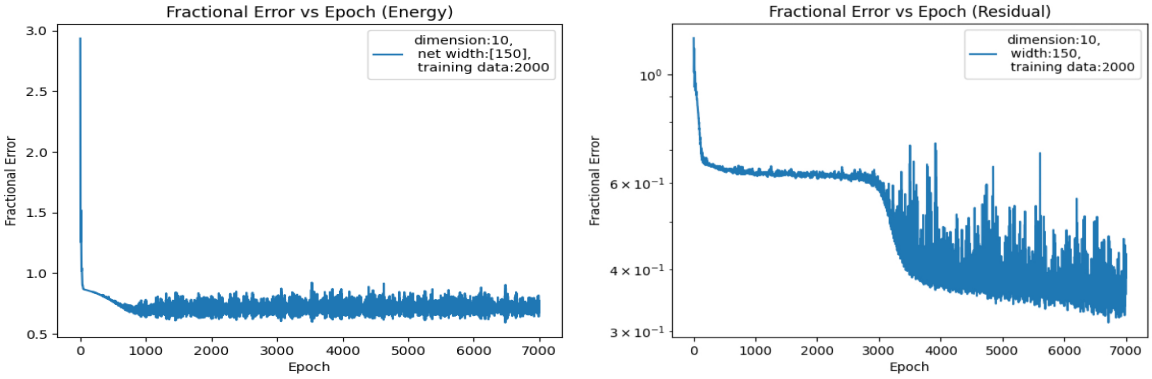


Figure 4: Error with epoch performance ($d = 10$). *Left* : $\hat{\mathcal{R}}_{\text{energy,loss-with-boundary-penalty}}$ (34). *Right*: $\hat{\mathcal{R}}_{\text{residual,loss-with-boundary-penalty}}$ (39).

4.2.1. This decision was made to investigate the impact of larger network widths on the performance of the methods and to exclude the possibility of improving high-dimensional performance by simply increasing the width of the network.

Table 1: Parameter configuration for different settings.

Loss Function	β	Learning Rate	d	Epoch	Net Width
$\hat{\mathcal{R}}_{\text{energy-b}(\lambda=1)}$(30)	0	0.0001	1, 2, 3 10	2000 7000	25, 50, 75, 100, 125, 150, 175, 200, 225, 250
$\hat{\mathcal{R}}_{\text{energy-b}(\lambda=5)}$(32)	0	0.0001	1, 2, 3 10	2000 7000	25, 50, 75, 100, 125, 150
$\hat{\mathcal{R}}_{\text{energy-p}}$(34)	50	0.0001	1, 2, 3 10	2000 7000	25, 50, 75, 100, 125, 150
$\hat{\mathcal{R}}_{\text{residual-b}(\lambda=1)}$(36)	0	0.0001	1, 2, 3 10	2000 7000	25, 50, 75, 100, 125, 150, 175, 200, 225, 250
$\hat{\mathcal{R}}_{\text{residual-b}(\lambda=5)}$(38)	0	0.0001	1, 2, 3 10	2000 7000	25, 50, 75, 100, 125, 150
$\hat{\mathcal{R}}_{\text{residual-p}}$(39)	200	0.0001	1, 2, 3 10	2000 7000	25, 50, 75, 100, 125, 150

4 Experiment Results

4.1 A comprehensive comparison of dimension 10 problem

Consider Equation (26). In higher dimensions, specifically $d = 10$, it has been observed that the six settings exhibit varying performance. A comprehensive analysis of the performance of these settings in terms of both empirical loss and fractional error at ten dimensions is provided in the subsequent discussion. In this section, the discussion is based on the dynamic of the loss function with respect to the epoch. It should be noted that the total network parameter is equal to $2w^2 + (d+4)w + 1 = 47101$ at this stage.

4.1.1 Boundary included model with trivial-scaling - variational energy and residual loss

Initially, the investigation focuses on exploring the performance of variational energy loss (Equation 30) and residual loss (Equation 36) with boundary included model (trivial-scaling, $\lambda = 1$). Based on the results obtained from the initial settings, it can be observed from Figure 5 that the variational energy loss and the residual loss attain convergence after about 4000 epochs. Further, Figure 6 indicates that both these methods yield similar performance in terms of fractional error, with the error being around 10^{-1}.

It is observable that the empirical loss for the energy method can attain negative values. *Nevertheless, in this preliminary configuration, the behavior of the two methods appears to be quite similar.*

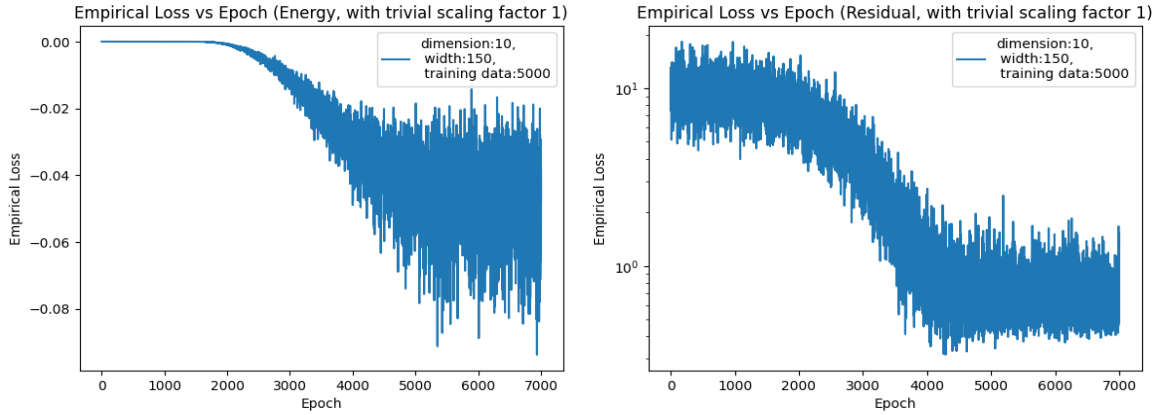


Figure 5: Empirical loss of different loss function ($d = 10$). *Left* : $\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=1}$ (30). *Right*: $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=1}$ (36).

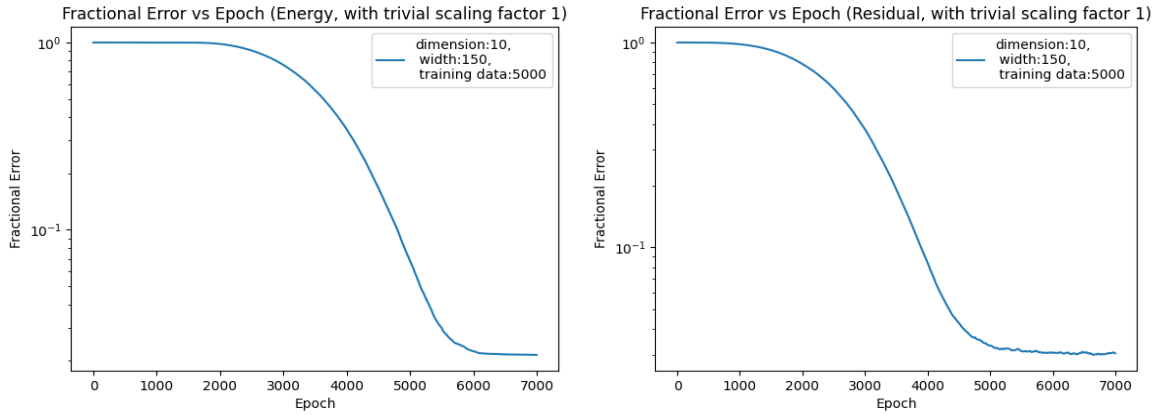


Figure 6: Fractional error of different loss function ($d = 10$). *Left* : $\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=1}$ (30). *Right*: $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=1}$ (36).

4.1.2 Boundary included model with non-trivial scaling - variational energy and residual loss

In this section the experiment compare the performance of variational energy loss (Equation 32) and residual loss (Equation 38) with boundary included model (non-trivial scaling, $\lambda = 5$). The introduction of the scaling factor resulted in a significant improvement in the performance of both methods. However, Figure 7 illustrates that the variational energy method (on the left) appears to be inadequately trained, while the residual method (on the right) yields better results. As can be seen from Figure 8 on the left, the fractional error for the variational energy method is found to be around 10^{-2}, while that of the residual method on the right is about 10^{-3}, indicating superior performance of the residual-based approach. In this particular setting, the empirical loss derived from variational energy demonstrates inferior performance when compared to the residual-based method. *The conclusions obtained in this section are similar to those in the previous section, in that the scaling factor $\lambda = 5$ is equivalent to expanding the output of the function, which facilitates the fitting of the neural network and enables it to achieve a higher fitting accuracy.*

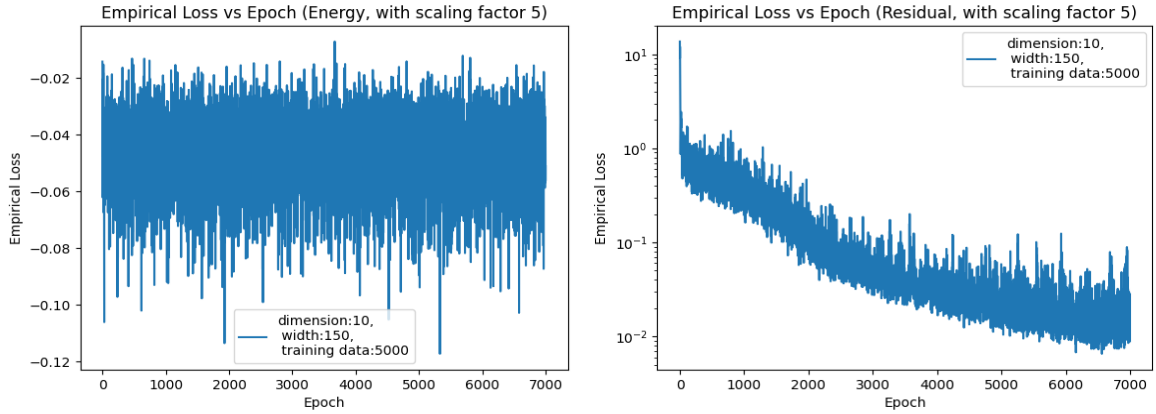


Figure 7: Empirical loss of different loss function ($d = 10$). *Left* : *inadequately – trained* – $\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=5}$ (32). *Right*: $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=5}$ (38).

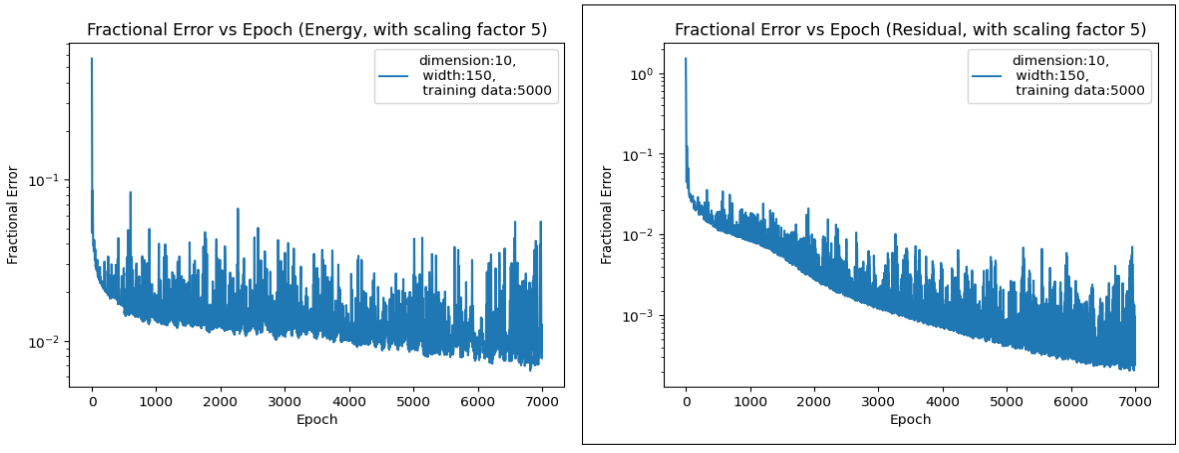


Figure 8: Fractional error of different loss function with best performance on the right ($d = 10$). *Left* : $\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=5}$ (32). *Right*: $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=5}$ (38).

4.1.3 Boundary penalty included - variational energy and residual loss

The experiment further explored the performance of the approach in satisfying boundary conditions in the variational energy method by incorporating a penalty term into the loss function (Equation 34), following Weinan's approach as described in (Weinan & Bing, 2017). The same setting was then applied to residual methods to evaluate their performance (Equation 39). In this specific problem, Figure 9 reveals that the variational methods (on the left) achieve a faster training speed, requiring only about 1000 epochs, whereas the residual method (on the right) necessitates 3000 epochs. From Figure 10, the left figure shows that the fractional error for variational energy methods oscillates around 60%, while the right figure indicates that the residual method hovers around 40%. *It has been observed that the residual loss exhibits slightly improved performance compared to the energy loss when a penalty term is added to the loss function.*

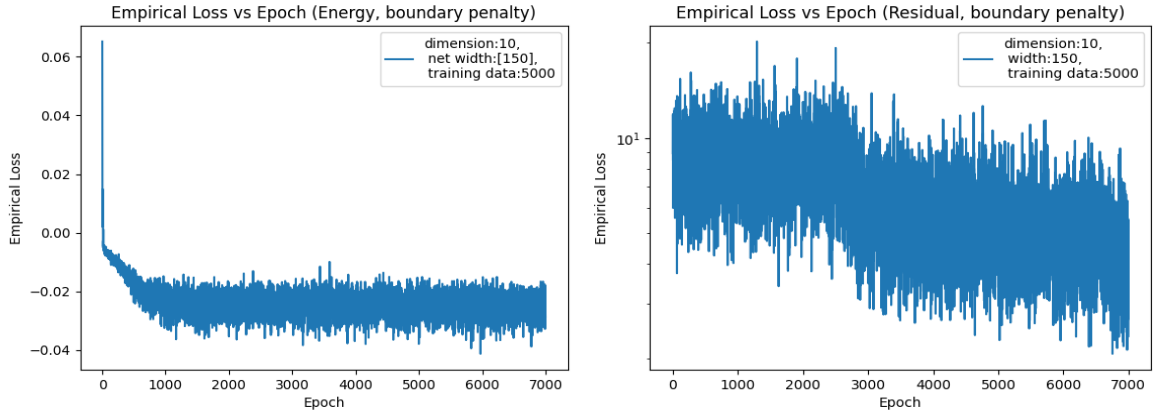


Figure 9: Empirical loss of different loss function ($d = 10$). *Left* : $\hat{\mathcal{R}}_{\text{energy,loss-with-boundary-penalty}}$ (34). *Right*: $\hat{\mathcal{R}}_{\text{residual,loss-with-boundary-penalty}}$ (39).

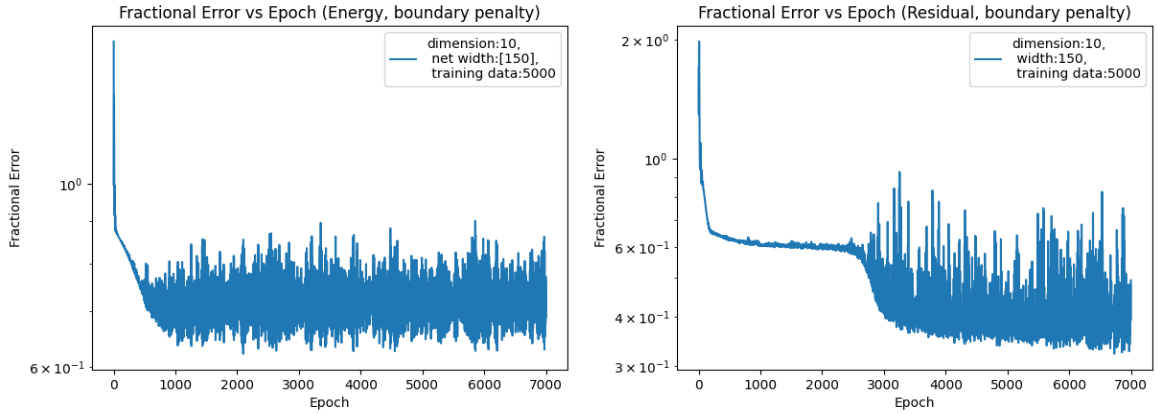


Figure 10: Fractional error of different loss function ($d = 10$). *Left* : $\hat{\mathcal{R}}_{\text{energy,loss-with-boundary-penalty}}$ (34). *Right*: $\hat{\mathcal{R}}_{\text{residual,loss-with-boundary-penalty}}$ (39).

4.1.4 Conclusion of dimension 10 problem

Through a comparative analysis of the performance of six distinct loss function configurations in ten dimensions, this experiment has identified the optimal combination: $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=5}$ *(38).* This combination yields the best global performance across all six settings under consideration, surpassing the setup of Equation 34, which follows Weinan's approach as described in the work of (Weinan & Bing, 2017).

4.2 Comparative analysis of dimension-dependent performance with different settings

To facilitate a clearer comparison of the performance of the different settings across different dimensions, the upcoming comparison will illustrate the performance of the various settings as the network width is varied with dimensionality, as opposed to the previous comparison of the loss function performance with respect to the epochs.

4.2.1 Dimension-dependent performance of boundary included model with trivial-scaling - variational energy and residual loss

In this section, the investigation focuses on exploring the performance of variable energy loss (Equation 30) and residual loss (Equation 36) with boundary included model (trivial scaling, $\lambda = 1$). The experiment commenced by employing an equivalent number of epochs and training samples for dimensions 1, 2, and 3. Subsequently, the number of epochs and training samples were augmented for dimension 10, followed by repeating the identical training process. The experiment first attempts to explore the network width

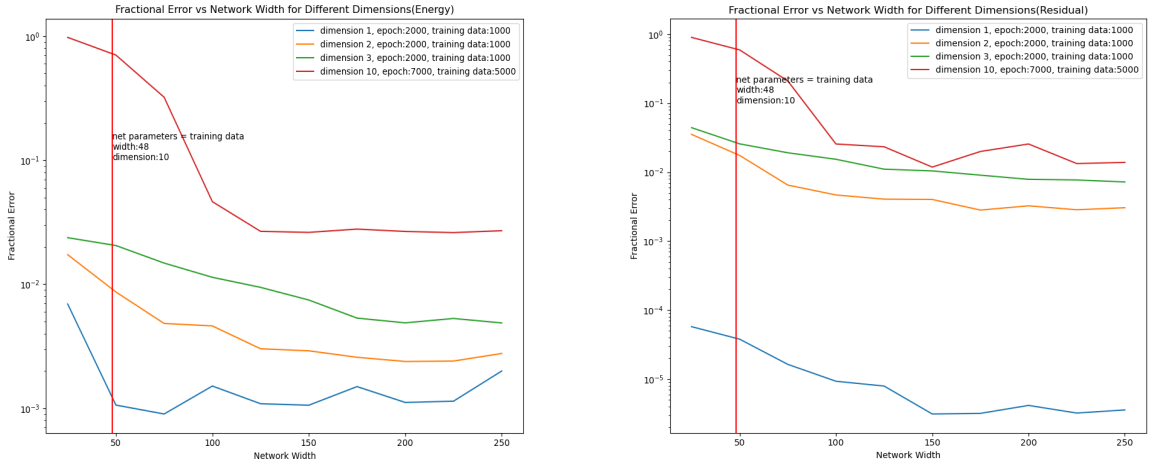


Figure 11: Degradation performance with increasing dimension with net width 250 ($d = 1, 2, 3, 10$). *Left* : $\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=1}$ (30). *Right*: $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=1}$ (36).

range from 25 to 150, which indicates that in a single dimension, the residual loss function exhibits superior performance in comparison to the energy loss function. And then the network width was raised to 250 while keeping all other parameters unchanged to investigate whether the fractional error in ten dimensions can be reduced to a similar level as in the lower dimensions by increasing the network width. Thus the total number of parameters is $2w^2 + (d+4)w + 1 = 128501$ when $d = 10$. As can be seen from Figure 11, the width is varying along the x axis, and the results revealed that the issue cannot be resolved solely by increasing the network width, the performance of the variational energy loss and the residual loss was found to decrease significantly with increasing dimensionality.

Specifically, the degradation performance with increasing dimension is consistent for the boundary-included model in terms of both variational energy and residual method. As can be seen from the Figure 11 on the left, the variational energy method demonstrates a gap between 10 and 123 dimensions. Then, Figure on the right shows that the residual method exhibits a significant gap between 1 and other dimensions. Overall, this indicates a similar trend in degradation performance as the dimension increases, regardless of whether it is the variational energy or residual loss being considered. To address this trend, the paper introduces a scaling factor in an attempt to improve the performance of the setting that satisfies the boundary conditions in the model.

4.2.2 Dimension-dependent performance of boundary included model with non-trivial scaling - variational energy and residual loss

In this section, the experiment aims to compare the performance of variational energy loss (Equation 32) and residual loss (Equation 38) in the context of a boundary-inclusive model with non-trivial scaling ($\lambda = 5$). Subsequently, the inclusion of a scaling factor resulted in a considerable enhancement in the performance of both energy and residual loss. The left figure demonstrates that the fractional error of the variational energy method reaches approximately 10^{-2} across all dimensions after setting *lambda* = 5. The right figure illustrates that as the network width increases, the fractional error of the residual method is around 10^{-5} in one, two, and three dimensions, and approximately 10^{-3} in 10 dimensions. *Based on Figure 12, it can be observed that although both methods exhibit performance gains after scaling, the improvement in residual loss is more pronounced.*

It is noteworthy that the performance of energy in 10 dimensions is nearly identical to that of 3 dimensions, whereas residual's performance in 10 dimensions outperforms its performance in lower dimensions ($d = 2, 3$).

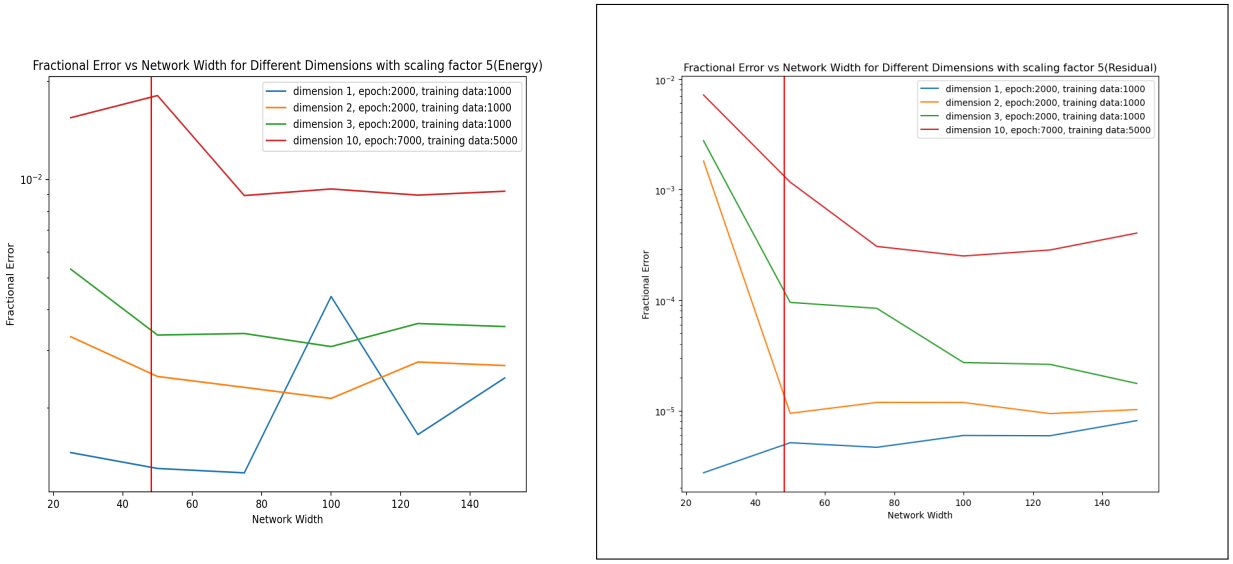


Figure 12: Fractional error of different loss function with scaling factor 5, with best performance on the right. *Left* : $\hat{\mathcal{R}}_{\text{energy,boundary-included-model},\lambda=5}$ (32). *Right*: $\hat{\mathcal{R}}_{\text{residual,boundary-included-model},\lambda=5}$ (38).

In the initial setup, the model that satisfied the boundary conditions exhibited diminishing output values as the dimensionality increased. This resulted in a significant increase in error at higher dimensions. *By adjusting the scaling factor to 5, the experiment normalizes the empirical loss at the beginning of training to approximately 1. This normalization exhibits a gradient increase with rising dimensionality. Through normalizing the output for each dimension, the overall performance experiences a substantial improvement.*

4.2.3 Dimension-dependent performance of loss with boundary penalty included - variational energy and residual loss

This experiment yielded observations regarding the performance of energy (Equation 34) and residual (Equation 39) in loss with penalty included. The resulting figure on the left is consistent with Weinan's approach as described in (Weinan & Bing, 2017).

In Figure 13, the left one presents the energy method, which displays greater consistency across different network widths. In one dimension, the error is observed to be approximately 10^{-2}. As the dimensions increase to two and three, the error experiences a substantial rise, reaching a value of 10^{-1}. Notably, the error exhibits an even more pronounced increase when evaluated in 10 dimensions. The right figure, on the other hand, showcases the residual method, which performs considerably better in dimension 1, reaching a value of 10^{-5}.

The penalty terms in the loss function, which are responsible for satisfying the boundary conditions, necessitate sampling at the boundary. As the dimensionality increases, the complexity of sampling rises linearly, potentially contributing to the heightened error observed in higher dimensions. *Overall, in this setting, the residual method demonstrates marginally superior performance compared to the energy method, particularly exhibiting strong performance in one dimension.*

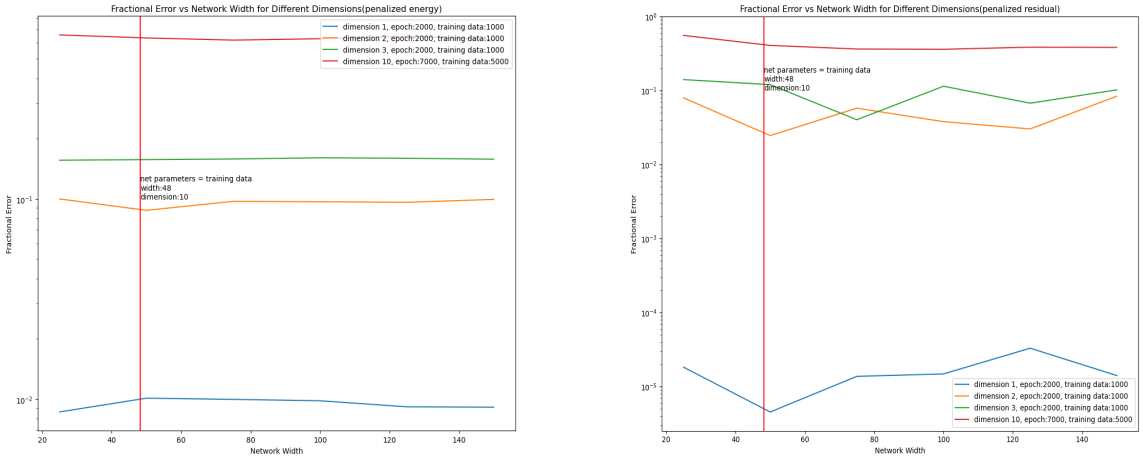


Figure 13: Fractional error of different loss function ($d = 1, 2, 3, 10$). *Left* : $\hat{\mathcal{R}}_{\text{energy,loss-with-boundary-penalty}}$ (34). *Right*: $\hat{\mathcal{R}}_{\text{residual,loss-with-boundary-penalty}}$ (39).

4.2.4 Overall conclusion

A total of six configurations were conducted in this experiment, and the corresponding number of neural networks trained for each configuration is provided below:

- Main Methods
  - Energy
    - Loss with Boundary Penalty, 40NNs (energy – p, 34)
    - Boundary Included Model
      - $\lambda = 1$, 24NNs (energy – b($\lambda = 1$), 30)
      - $\lambda = 5$, 24NNs (energy – b($\lambda = 5$), 32)
  - Residual
    - Loss with Boundary Penalty, 40NNs (residual – p, 39)
    - Boundary Included Model
      - $\lambda = 1$, 24NNs (residual – b($\lambda = 1$), 36)
      - $\lambda = 5$, 24NNs (residual – b($\lambda = 5$), 38)

Initially, in the setting of satisfying the boundary conditions within the model, this experiment sought to minimize the error by increasing the network width. Ten distinct network widths (up to 250) were tested for both the energy and residual methods in this configuration, resulting in the training of 40 neural networks across four dimensions in total. However, upon discovering that merely augmenting the network width yielded no significant improvement, the initial six network widths (up to 150) were employed for all subsequent configurations. Consequently, the training was conducted with 24 neural networks. Thus, a total number of 176 neural networks were trained for this experiment.

Since the approach of adding a penalty term to the variational energy loss function ($\hat{\mathcal{R}}_{\text{energy-p}}$, 34) is based on the work of Weinan & Bing (2017), the experiment further investigates the performance comparison between satisfying boundary conditions in the model and satisfying boundary conditions in the loss function for the variational energy approach, specifically in relation to this particular PDE problem. This comparison enables a more comprehensive understanding of the effectiveness of each method.

From Figures 12 and 13 on the left, it is clear that there is minimal change in the fluctuation of the penalized energy loss with respect to variations in network width. Additionally, for the variational energy loss with a penalty term, in dimensions 2 and 3, the fractional error only reaches around 10%. However, in the case of ten dimensions, the fractional error remains at around 63%. In comparison, satisfying boundary conditions in the model performs better in all dimensions.

Nevertheless, the variational energy loss with a penalty term yields a more rapid convergence, as evidenced by the Figure 5 and 9, where a superior fit is attained at approximately 1000 epochs in ten dimensions, while the setting of bounded in model attains a comparable outcome at approximately 5000 epochs.

For a more comprehensive comparison, the study focused on comparing the performance of different settings to satisfy the boundary conditions across various network widths. Specifically, the performance at a network width of 150 was chosen for comparison, as this width showed the best performance in terms of fractional error for all the settings that satisfy the boundary conditions in the model.

Therefore, Table 2 presents the quantitative comparison over all six settings. The table clearly shows that Residual $-$ b($\lambda = 5$) delivers superior results in all categories, surpassing the performance of other settings.

Table 2: Error with different method, width = 150.

d	$\mathcal{E}_{\text{energy-p}}$	$\mathcal{E}_{\text{energy-b}(\lambda=1)}$	$\mathcal{E}_{\text{energy-b}(\lambda=5)}$	$\mathcal{E}_{\text{residual-p}}$	$\mathcal{E}_{\text{residual-b}(\lambda=1)}$	$\mathcal{E}_{\text{residual-b}(\lambda=5)}$
1	0.0091	0.0015	0.0025	1.4111×10^{-5}	3.6318×10^{-6}	$8.1282e^{-6}$
2	0.0994	0.0032	0.0027	0.0842	0.0015	1.0232×10^{-5}
3	0.1578	0.0075	0.0036	1.1028	0.0111	1.7623×10^{-5}
10	0.6245	0.0212	0.0095	0.5138	0.0223	0.0004

After conducting a comprehensive evaluation of several techniques, it is evident that the combination of residual, boundary included model and scale techniques ($\hat{\mathcal{R}}_{\text{residual-b}(\lambda=5)}$, 38), i.e., $residual + boundary - included - model + scaling factor 5$, exhibits superior performance in this particular test, produce best performance in every dimensions.

Simultaneously, it has been observed that the variational energy method, incorporating a loss function with a penalty term ($\hat{\mathcal{R}}_{\text{energy-p}}$, 34), yields the most rapid convergence rate, albeit not achieving the optimal performance. And it is noteworthy that both types of energy method exhibit considerably faster training compared to residual method.

In addition to the exploration of these six settings, the study also uncovered limitations when dealing with really higher-dimensional problems. The experiment was conducted on the same Poisson equation (26), extending it up to 50 dimensions. The results indicated that training reliability consistently decreased as the problem's dimensionality increased. All three configurations of the energy loss function exhibited poor training performance, while the residual loss function required a considerably longer training time as the dimensionality grew.

Several possible reasons for these limitations merit further discussion. One potential explanation is the curse of dimensionality, which negatively impacts the performance of deep learning models when dealing with high-dimensional problems. Another possible reason could be the increased complexity of the PDEs as dimensions increase, making it more challenging for the neural network to learn the underlying patterns and relationships. Finally, the optimization process for high-dimensional problems can be more difficult, as the search space becomes exponentially larger, which may lead to the model converging to suboptimal solutions. Further research is needed to address these challenges and improve the performance of neural network-based methods for solving high-dimensional PDEs.

However, it is crucial to note that the optimal approach may vary depending on the specific problem being addressed. Therefore, further experimentation may be required to determine the most appropriate approach for a given task.

5 Summary and Outlook

5.1 Project Work Summary

The Deep Ritz Method (DRM) is a technique that uses variational energy loss for solving partial differential equations (PDEs) and employs boundary penalty terms or boundary-included models to enforce boundary conditions. By comparing DRM with other approaches, such as the Deep Galerkin Method (DGM), this study aims to identify the most effective method for solving the specific Poisson PDE.

Experimental results demonstrate the necessity of combining elements from various methods to achieve optimal performance in PDE solutions. The modification method, which incorporates residual minimization, boundary-included models, and an additional scaling factor, exhibits superior performance in solving the Poisson PDE compared to other individual techniques.

Solving PDEs necessitates the use of very deep networks, as they capitalize on a greater amount of contextual information and employ multiple nonlinear layers to model intricate functions. The experiments clearly demonstrate that deep networks outperform their shallow counterparts. The variational energy method converges at a significantly faster pace compared to residual one, albeit with lower accuracy. To tackle the challenges associated with sampling at high-dimensional spatial boundaries, researchers have explored various approaches, including the utilization of Monte Carlo methods (Chen et al., 2021). This direction holds potential for future developments in addressing such complexities.

The proposed modification method's (with multiplicative regularizer) reduces the error in each dimension to almost a level in dimension-dependent problems. This approach effectively reduces the model capacity (number of parameters) of the network, decreases the number of models, and obtains a more general model for solving PDEs using deep learning techniques. Despite these advantages, the limitations associated with extremely high-dimensional problems still need to be considered. Further research efforts should focus on addressing these challenges to improve the overall efficacy of deep learning approaches in solving high-dimensional partial differential equations.

5.2 Possible future developments

In recent years, deep learning methods applied to solving complex differential equations in engineering applications, financial phenomena, and the recent emergence of the spread of the novel coronavirus pandemic have yielded substantial results. Based on the report writing and a survey of related literature, it is posited that the deep learning solutions for differential equations will likely have the following potential or feasible research directions:

- Solving and predicting complex high-dimensional differential equations (dynamical systems): Although related deep learning methods have achieved satisfactory results for some typical differential equation solving problems, there is no widely accepted method for addressing high-dimensional, complex problems. Han et al. (2017) and Sirignano & Spiliopoulos (2017) attempted time-based data discretization methods for segmental approximation to solve high-dimensional partial differential equations. However, there are several theoretical limitations, and the data handled are relatively simple. This issue will become increasingly significant as the demand for applications grows.

- Improvement of solving/prediction accuracy and training efficiency: Since deep learning methods utilize numerical methods with nonlinear function approximation instead of analytical methods, approximation errors in the network always exist in theory and practical applications (Hornik et al., 1989). Consequently, reducing the approximation error to fit various application scenarios remains a primary research focus. Increasing the network complexity is a feasible idea (Sirignano & Spiliopoulos, 2017; Meade & Fernandez, 1994), but as the network complexity grows, so do the time and space complexities of training. Overly large neural networks may not be practical in real-world applications.

- Generalizability of neural network approximate solutions: Within a given definition domain, when the training data is reasonably distributed and sufficiently available, a neural network can achieve a good fit through proper design and training. However, the performance of the neural network is often not as satisfactory outside the given data set. In some cases, the network performs well on the training set but poorly on the test set, a phenomenon known as "overfitting." Generalization remains a key research focus in neural network applications. Enhancing network complexity and increasing data coverage are effective methods, but they also entail significant computational demands. Training a large network for each task would likely result in redundancy.

- Integration of multimodal data in applications: As the potential of deep learning methods for solving differential equations continues to be explored, applying these methods to complex real-world systems or investigating more intricate scientific problems is a natural progression for research. Unlike the numerical data typically handled by differential equation solutions, practical applications often encounter more complex situations and typically involve multimodal data, such as images, videos, and natural languages. The attempt to describe the morphodynamics of cell-drug interactions based on images in Cavanagh et al. (2021) represents a preliminary exploration towards the practical application of such methods. However, designing multimodal data processing methods from a more macroscopic perspective and accelerating their practical application remain outstanding challenges.

- Incorporation of complex social factors in practical applications: In theoretical studies and practical applications of differential equations and other scientific problems, the individuals who participate in, explore, and solve these scientific problems bring with them inescapable social factors that cannot be ignored. Consequently, determining how to represent human behavior and take human influence and evaluation into account in differential equations is a topic worth investigating. However, existing analytical methods and frameworks often overlook social factors, which hinders the further development of these methods. Thus, there is an urgent need to establish a comprehensive theoretical framework for exploring and analyzing social factors in complex systems and effectively integrating these factors into solutions.

- Solving differential equations (or predicting dynamical systems) using deep learning essentially leverages the nonlinear function approximation capabilities of neural networks to approximate differential equation solutions. In general, partial differential equations have analytical solutions composed of various functional forms, such as power functions, exponential functions, and logarithmic functions, which can be considered as basis functions in Galerkin's method. As a result, the solutions of the same differential equation (or different differential equations) under different parameters can be composed of the same basis function with different parameters (or different basis functions and different parameters). Therefore, in practical research, training a network for each differential equation individually is undoubtedly a time-consuming and unnecessary task. Simultaneously, in practical applications, as the complexity of deep learning models increases, the model performance and generalization also improve (Sirignano & Spiliopoulos, 2017; Meade & Fernandez, 1994). Furthermore, the control equations of physical systems in specific application scenarios are similar, suggesting that constructing oversized neural network models to solve the differential equations of complex dynamical systems could be an effective approach in the future.

In the following analysis, the potential of utilizing large models for solving differential equations is explored. In recent years, the success of large models in natural language processing has gained widespread attention, with their generalization ability and capacity to quickly adapt to downstream tasks garnering recognition from researchers, industries, and markets. Devlin et al. (2019) proposed a large natural language processing model called BERT, which achieved the best performance at that time in 11 different natural language tests and ushered in the era of AI large models. Brown et al. (2020) introduced GPT-3, the best-performing model in the GPT family, which has 175 billion parameters and improved generalization. Radford et al. (2021) proposed a large-scale visual-linguistic pre-training model called CLIP, which uses contrast learning for training; by predicting whether images and text match in a pre-trained task, CLIP can adapt to various image and language downstream tasks. Alayrac et al. (2022) proposed Flamingo, a visual-linguistic model for small sample problems that outperforms other related large models. Su et al. (2022) introduced large models to the biological domain and proposed a multimodal analysis large model. Xie et al. (2021) proposed an

image macro-model based on masked image modeling (MIM) called SimMM, which can reduce computational effort. Large model theories and applications have become a research hotspot in the artificial intelligence field, with more descriptions of large models found in Bommasani et al. (2021).

In addition to the aforementioned large models, the recent large model ChatGPT has achieved significant success in tasks such as text processing, dialogue systems, and image description. Consequently, there are reasons to believe that large models may contribute to progress in solving differential equations in the future.

References

Jean-Baptiste Alayrac, Jeff Donahue, Pauline Luc, Antoine Miech, Iain Barr, Yana Hasson, Karel Lenc, Arthur Mensch, Katie Millican, Malcolm Reynolds, Roman Ring, Eliza Rutherford, Serkan Cabi, Tengda Han, Zhitao Gong, Sina Samangooei, Marianne Monteiro, Jacob Menick, Sebastian Borgeaud, Andy Brock, Aida Nematzadeh, Sahand Sharifzadeh, Mikolaj Binkowski, Ricardo Barreira, Oriol Vinyals, Andrew Zisserman, and Karen Simonyan. Flamingo: a visual language model for few-shot learning. *ArXiv*, abs/2204.14198, 2022.

Georgios D. Barmparis and Giorgos P. Tsironis. Physics-informed machine learning for the covid-19 pandemic: Adherence to social distancing and short-term predictions for eight countries. *Quant. Biol.*, 10: 139, 2020.

Christian Beck, Sebastian Becker, Philipp Grohs, Nor Jaafari, and Arnulf Jentzen. Solving the kolmogorov pde by means of deep learning. *Journal of Scientific Computing*, 88, 2018.

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.

Richard Bellman. *Dynamic Programming*. Princeton Landmarks in Mathematics. Princeton University Press, Princeton, NJ, 2010. Reprint of the 1957 edition, with a new introduction by Stuart Dreyfus.

Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018.

Hao bin Shi and Max Qinghu Meng. Deep koopman operator with control for nonlinear systems. *IEEE Robotics and Automation Letters*, 7:7700–7707, 2022.

Rishi Bommasani, Drew A Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, et al. On the opportunities and risks of foundation models. *arXiv preprint arXiv:2108.07258*, 2021.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, T. J. Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeff Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners. *ArXiv*, abs/2005.14165, 2020.

Ben Calderhead, Mark Girolami, and Neil Lawrence. Accelerating bayesian inference over nonlinear differential equations with gaussian processes. *Advances in neural information processing systems*, 21, 2008.

Henry Cavanagh, Andreas Mosbach, Gabriel Scalliet, Rob Lind, and Robert G. Endres. Physics-informed deep learning characterizes morphodynamics of asian soybean rust disease. *Nature Communications*, 12, 2021.

Jingrun Chen, Rui Du, Panchi Li, and Liyao Lyu. Quasi-monte carlo sampling for solving partial differential equations by deep neural networks. *Numerical Mathematics*, 14:377–404, 2021.

T Ciodaro, D Deva, JM De Seixas, and D Damazio. Online particle detection with neural networks based on topological calorimetry information. In *Journal of physics: conference series*, volume 368, pp. 012030. IOP Publishing, 2012.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Jean Donea and Antonio Huerta. *Finite element methods for flow problems*. John Wiley & Sons, 2003.

L.C. Evans. *Partial Differential Equations*. Graduate studies in mathematics. American Mathematical Society, 1998. ISBN 9780821807729. URL https://books.google.co.uk/books?id=5Pv4LVB_m8AC.

Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88:303–338, 2010.

MD Feit, JA Fleck Jr, and A Steiger. Solution of the schrödinger equation by a spectral method. *Journal of Computational Physics*, 47(3):412–433, 1982.

Taniel S. Franklin, Leonardo da Silva Souza, Raony M. Fontes, and Márcio André Fernandes Martins. A physics-informed neural networks (pinn) oriented approach to flow metering in oil wells: an esp lifted oil well system as a case study. *Digital Chemical Engineering*, 2022.

Notes Garbled. 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.

Kathrin Glau and Linus Wunderlich. The deep parametric pde method and applications to option pricing. *Appl. Math. Comput.*, 432:127355, 2022.

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.

Somdatta Goswami, Cosmin Anitescu, Souvik Lal Chakraborty, and Timon Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *ArXiv*, abs/1907.02531, 2019.

Jiequn Han, Arnulf Jentzen, and Weinan E. Solving high-dimensional partial differential equations using deep learning. *Proceedings of the National Academy of Sciences*, 115:8505 – 8510, 2017.

Jane K. Hart and Kirk Martinez. Environmental sensor networks: A revolution in the earth system science? *Earth-Science Reviews*, 78(3):177–191, 2006. ISSN 0012-8252. doi: https://doi.org/10.1016/j.earscirev.2006.05.001. URL https://www.sciencedirect.com/science/article/pii/S0012825206000511.

Alexander Heinlein, Axel Klawonn, Martin Lanser, and Janine Weber. Combining machine learning and domain decomposition methods for the solution of partial differential equations—a review. *GAMM-Mitteilungen*, 44(1):e202100001, 2021.

Moritz Helmstaedter, Kevin L Briggman, Srinivas C Turaga, Viren Jain, H Sebastian Seung, and Winfried Denk. Connectomic reconstruction of the inner plexiform layer in the mouse retina. *Nature*, 500(7461):168–174, 2013.

Kurt Hornik, Maxwell B. Stinchcombe, and Halbert L. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2:359–366, 1989.

Ameya D Jagtap and George E Karniadakis. Extended physics-informed neural networks (xpinns): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations. In *AAAI Spring Symposium: MLPS*, pp. 2002–2041, 2021.

John Jumper, Richard Evans, Alexander Pritzel, Tim Green, Michael Figurnov, Olaf Ronneberger, Kathryn Tunyasuvunakool, Russ Bates, Augustin dek, Anna Potapenko, et al. Highly accurate protein structure prediction with alphafold. *Nature*, 596(7873):583–589, 2021.

George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.

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

Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE transactions on neural networks*, 9(5):987–1000, 1998.

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

Randall J. LeVeque. *Finite Difference Methods for Ordinary and Partial Differential Equations*. Society for Industrial and Applied Mathematics, 2007. doi: 10.1137/1.9780898717839. URL https://epubs.siam.org/doi/abs/10.1137/1.9780898717839.

Ke Li, Kejun Tang, Tianfan Wu, and Qifeng Liao. D3m: A deep domain decomposition method for partial differential equations. *IEEE Access*, 8:5283–5294, 2019a.

Wuyang Li, Xueshuang Xiang, and Yingxiang Xu. Deep domain decomposition method: Elliptic problems. In *Mathematical and Scientific Machine Learning*, pp. 269–286. PMLR, 2020.

Xuan Li, Yutong Wang, Lan Yan, Kunfeng Wang, Fang Deng, and Fei-Yue Wang. Paralleleye-cs: A new dataset of synthetic images for testing the visual intelligence of intelligent vehicles. *IEEE Transactions on Vehicular Technology*, 68(10):9619–9631, 2019b.

Gui-Rong Liu and Yuan-Tong Gu. *An introduction to meshfree methods and their programming*. Springer Science & Business Media, 2005.

Zichao Long, Yiping Lu, Xianzhong Ma, and Bin Dong. Pde-net: Learning pdes from data. In *International conference on machine learning*, pp. 3208–3216. PMLR, 2018.

Zichao Long, Yiping Lu, and Bin Dong. Pde-net 2.0: Learning pdes from data with a numeric-symbolic hybrid deep network. *Journal of Computational Physics*, 399:108925, 2019.

Andrew J. Meade and A. A. Fernandez. Solution of nonlinear ordinary differential equations by feedforward neural networks. *Mathematical and Computer Modelling*, 20:19–44, 1994.

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

Mohammad Amin Nabian and Hadi Meidani. A deep learning solution approach for high-dimensional random differential equations. *Probabilistic Engineering Mechanics*, 57:14–25, 2019.

Julian D. Osorio, Zhicheng Wang, George E. Karniadakis, Shengze Cai, Chrys Chryssostomidis, Mayank Panwar, and Rob Hovsapian. Forecasting solar-thermal systems performance under transient operation using a data-driven machine learning approach based on the deep operator network architecture. *Energy Conversion and Management*, 2022.

Guofei Pang, Lu Lu, and George Em Karniadakis. fpinns: Fractional physics-informed neural networks. *SIAM Journal on Scientific Computing*, 41(4):A2603–A2626, 2019.

Richard H Pletcher, John C Tannehill, and Dale Anderson. *Computational fluid mechanics and heat transfer*. CRC press, 2012.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*, 2021.

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, 2019.

Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations. *Science*, 367(6481):1026–1030, 2020.

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.

David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *nature*, 550(7676):354–359, 2017.

Justin Sirignano and Konstantinos Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *Journal of computational physics*, 375:1339–1364, 2018.

Justin A. Sirignano and Konstantinos V. Spiliopoulos. Dgm: A deep learning algorithm for solving partial differential equations. *J. Comput. Phys.*, 375:1339–1364, 2017.

Bing Su, Dazhao Du, Zhao-Qing Yang, Yujie Zhou, Jiangmeng Li, Anyi Rao, Haoran Sun, Zhiwu Lu, and Ji rong Wen. A molecular multimodal foundation model associating molecule graphs with natural language. *ArXiv*, abs/2209.05481, 2022.

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.

E Weinan and Yu Bing. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems, 2017.

Keke Wu, Rui Du, Jingrun Chen, and Xiang Zhou. Understanding loss landscapes of neural network models in solving partial differential equations, 2021.

Zhenda Xie, Zheng Zhang, Yue Cao, Yutong Lin, Jianmin Bao, Zhuliang Yao, Qi Dai, and Han Hu. Simmim: a simple framework for masked image modeling. *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 9643–9653, 2021.

Neha Yadav, Anupam Yadav, Manoj Kumar, et al. *An introduction to neural network methods for differential equations*, volume 1. Springer, 2015.

Yibo Yang and Paris Perdikaris. Adversarial uncertainty quantification in physics-informed neural networks. *Journal of Computational Physics*, 394:136–152, 2019.

Dongkun Zhang, Lu Lu, Ling Guo, and George Em Karniadakis. Quantifying total uncertainty in physics-informed neural networks for solving forward and inverse stochastic problems. *Journal of Computational Physics*, 397:108850, 2019.

Enrui Zhang, Minglang Yin, and George Em Karniadakis. Physics-informed neural networks for nonhomogeneous material identification in elasticity imaging. *ArXiv*, abs/2009.04525, 2020.

Yinhao Zhu, Nicholas Zabaras, Phaedon-Stelios Koutsourelakis, and Paris Perdikaris. Physics-constrained deep learning for high-dimensional surrogate modeling and uncertainty quantification without labeled data. *Journal of Computational Physics*, 394:56–81, 2019.

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
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
$\mathrm{diag}(\boldsymbol{a})$	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 A
$\mathsf{A}_{:,:,i}$	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

$f : \mathbb{A} \to \mathbb{B}$	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, $\log(1 + \exp(x))$
$\left\|\boldsymbol{x}\right\|_p$	L^p norm of $\boldsymbol{x}$
$\left\|\boldsymbol{x}\right\|$	L^2 norm of $\boldsymbol{x}$
x^+	Positive part of x, i.e., $\max(0, x)$
$\mathbf{1}_{\text{condition}}$	is 1 if the condition is true, 0 otherwise
$\arg\max_{\mathbf{x} \in \mathbb{R}^n} f$	
$\arg\min_{\mathbf{x} \in \mathbb{R}^n} f$	
$\lambda, \text{rectifier}, \text{softmax}, \sigma, \zeta, \text{Var}, \text{SE}$	