Solving Navier-Stokes Equations in Protoplanetary Disk Using Physics-informed Neural Networks

by

Shunyuan Mao
B.Sc., University of Science and Technology of China, 2019

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Physics and Astronomy

Solving Navier-Stokes Equations in Protoplanetary Disk Using Physics-informed Neural Networks

by

Shunyuan Mao
B.Sc., University of Science and Technology of China, 2019

Supervisory Committee

Dr. Ruobing Dong, Supervisor
(Department of Physics and Astronomy)

Dr. Kwang Moo Yi, Outside Member
(Department of Computer Science at the University of British Columbia)

ABSTRACT

We show how physics-informed neural networks can be used to solve compressible Navier-Stokes equations in protoplanetary disks. While young planets form in protoplanetary disks, because of the limitation of current techniques, direct observations of them are challenging. So instead, existing methods infer the presence and properties of planets from the disk structures created by disk-planet interactions. Hydrodynamic and radiative transfer simulations play essential roles in this process. Currently, the lack of computer resources for these expensive simulations has become one of the field's main bottlenecks. To solve this problem, we explore the possibility of using physics-informed neural networks, a machine learning method that trains neural networks using physical laws, to substitute the simulations. We identify three main bottlenecks that prevent the physics-informed neural networks from achieving this goal, which we overcome by hard-constraining initial conditions, scaling outputs and balancing gradients. With these improvements, we reduce the relative $\mathbb{L}_2$ errors of predicted solutions by $97\% \sim 99\%$ compared to the vanilla PINNs on solving compressible Navier-Stokes equations in protoplanetary disks.

Contents

List of Tables

List of Figures

ACKNOWLEDGEMENTS

I would like to thank:

My family, for supporting me.

Ruobing Dong, for mentoring, support, encouragement, and patience.

Kwang Moo Yi, Lu Lu, Sifan Wang, Paris Perdikaris, Xiaowei Jin, for suggestions on the project.

DEDICATION

To my family, my teachers, my friends and my supervisors.

Chapter 1

Introduction

1.1 Protoplanetary Disk

In the past few decades, exoplanet surveys using multiple techniques have revealed that planets are essentially ubiquitous in the universe [3, 4]. They form in "protoplanetary disk", which are flattened rotating disks surrounding newborn stars that typically last for a few million years. How these planets and the Earth form is one of the most thrilling questions in astronomy.

Ideally, the most straightforward way to study planet formation is to observe samples of forming planets in different stages. However, it has been extremely challenging to detect forming planets directly because they are tiny and faint compared to the host stars and disks. So far, only a handful of such planets have been detected [5, 6].

Fortunately, forming planets can be identified through the disk structures they produced via gravitational disk-planet interactions. These structures are much larger and brighter than the planets themselves and can be easily identified in observations today. In fact, over past decades, such structures have been found in hundreds of disks, and compelling evidence has shown that many of them are likely produced by disk-planet interactions [7, 8]. Combining the theory of disk-planet interactions and simulations, we can infer the presence of planets in these disks and further constrain their masses and orbits, which is crucial for understanding planet formation.

To identify and infer the properties of forming planets from the disk structures, the typical modeling process is as follows:

1. Make an experience-based initial guess on the planetary configuration.

2. Run hydrodynamic and radiative transfer simulations to generate synthetic ob-

servations for the initial guess.

3. Compare the synthetic observations with real observations, identify differences and update the models by hand.

4. Repeat the steps 2 and 3 above (usually at least tens of rounds) until an acceptable fit is achieved.

The step 2 is the most computationally expensive step, which involves solving partial differential equations (PDEs) using numerical solvers. Typically, modeling a single system consumes up to a million CPU hours.

The computational burden arising from step 2 is a problem, since disk observations are exploding. In the past several years, with the latest generation of telescopes such as ALMA and VLT/SPHERE, a few hundred disks have been observed. As more instruments are still being developed and commissioned (e.g., SCExAO on Subaru), high resolution and high sensitivity disk observations will be carried out at an ever-faster pace. Among all these disks that have been observed, only a dozen systems have been modeled. To process all of them, hundreds of millions of CPU hours are needed. To put this into perspective, this typically costs on the order of 10 million USD at commercial services such as Amazon HPC. This is clearly impractical.

Therefore, there is an urgent need for a new method to solve PDEs in protoplanetary disks, model the disk-planet interactions, and generate hundreds of synthetic observations with less computational cost than traditional numerical solvers.

1.2 Previous Work

Various alternatives to traditional numerical solvers that solve PDEs with less computational resources and time have been explored. Among them, machine learning methods have gained popularity in recent years thanks to their success in many different application areas [9, 10, 11].

For example, data-driven machine learning methods have been proposed to infer the presence and constrain the properties of hidden planets [12, 13]. They build neural networks whose inputs are measurable disk morphology parameters [12] or synthetic disk images [13], while the outputs are the predicted hidden planet masses. To train the neural networks, labeled data on a large enough number of protoplanetary disks is necessary to cover the disk-parameter space. However, there are still very few

disks from real observations that have planet masses measured. Thus, they used hydrodynamic simulations to generate simulated disks as labeled data. In fact, to generate the data sets, they run 1200 simulations that evolve around 2000 orbits [13]. As a result, though they get promising results, it is still highly computationally expensive, making extending the method to more complex disk systems challenging. In addition, with 1200 simulations, only a small parameter space has been explored, making it difficult for the method to be applied to real disk observations.

Instead of using simulations to synthesize labeled data, another promising way of overcoming the lack of training data is using information from physical laws to constrain the neural network while training. This is generally named physics-informed (or physics-constrained) machine learning [14, 15, 16, 17]. The method is widely applicable to problems whose physical laws are fully or partially known. The main idea is to use the information of physical laws to teach neural networks. By allowing physical laws to participate in the training, the dependence on labeled data can be greatly reduced.

Recently, among the physics-informed methods, physics-informed neural networks (PINNs) [14], has grown rapidly and has shown great potentials for applications in many fields [18, 19, 20, 21, 22]. The method has shown promising potentials on two kinds of PDE problems:

Forward problem Given a complete set of PDEs, initial conditions and boundary conditions, we want the solutions of the equations [14, 21].

Inverse problem Given part of the PDEs and additional information about the solutions, reveal the unknowns in the equations [14]. For example, in some cases, one or more PDE coefficients might be unknown, but the PDE solutions are available, and the aim is to constrain the unknown coefficients from the solutions [19, 21].

The method's main idea is to incorporate the physical laws into the loss function in the form of mathematical equations. The neural network (NN) then performs as a function approximator of the solution. The inputs to the NN are the known variables in the equations, and the outputs are the predicted solutions. The loss functions are composed of physical laws and labeled data. In training, the loss function penalizes the NN and biases it towards the solution of the equations. In the forward problem, the trainables are the NN parameters. When the loss is low enough, the network is a close approximation of the real solution of the problem. In the inverse

problem, trainables also include the unknowns in the PDEs. When the neural network is trained, the optimized trainables reveal the unknown PDE coefficients. To evaluate the mathematical equations in the loss functions, the partial derivatives of the unknown variables are needed, which are equivalent to the gradients of the neural networks' outputs with respect to the inputs. They carefully design the neural networks to evaluate the partial derivatives using the same automatic differentiation technique as the backward propagation method widely used in deep learning.

Previous works have shown PINNs are powerful and efficient in solving forward problems ranging from Poisson equations [23], Laplace equations [23], 1D and 2D Burgers' equations [14] to incompressible Navier-Stokes equations [18]. Works on solid mechanics [24] and fluid dynamics [21] also have shown PINNs' potentials in inverse problems.

Although PINNs have been shown to have great potential and have received wide attention recently, there are still several drawbacks:

Gradients-balancing problem The back-propagated gradients are not balanced in the training process. The loss functions of PINNs are usually composed of several terms: residuals of PDEs, errors on initial conditions, errors on boundary conditions, and errors on observed data. The gradients with respect to the NN parameters, derived from these loss terms respectively, may numerically differ by several orders of magnitude [25], [26]. Therefore, if we simply sum these gradients up to yield the direction of updating the NN parameters, a few large terms would dominate the training and prevent the neural networks from converging to global optima.

Model-generalization problem For the method in [14] (hereafter vanilla PINNs), one trained neural network only represents the solution of one PDE problem. Any changes of the problem, such as changes of the value of PDE coefficients or changes of initial conditions, require fresh re-training of the neural networks. Since training PINNs usually requires much longer time and more computational resources than running a numerical solver, the method is actually less efficient than numerical methods. As a result, improvements are needed for it to be a candidate of PDE solvers.

Long-time-integration problem In time-dependent problems, the errors grow as the evolution time [18, 27], preventing us from applying the vanilla PINNs to problems with long time intervals.

Below, we will go through the available or potential solutions for these drawbacks.

Generally, there are two ways to our knowledge to ease the gradients-balancing problem, though there is no known method to eliminate it. The first way is to reduce the number of loss terms by satisfying the constraints naturally, without training [28, 23]. For example, we will show in Section 4.3.1 and Section A.1 that by modifying the neural network architecture, we can reduce the number of loss terms to minimize, thus easing the conflicts between the gradients of loss terms. Another efficient way is to balance the contributions of the gradients. Adding and properly adjusting weights for gradients can efficiently balance them. Then the problem now becomes how one can properly set the weights. We will show in Section 4.3.3 that the value of loss terms at the initial training step is a suitable option to determine the weights. Meanwhile, previous work also proposed using the convergence rates of errors to estimate the weights [26]. Furthermore, they show that Neural Tangent Kernel [29] Matrix can effectively estimate the convergence rates. We examine both methods as well and discussed the results in Section 4.3.3.

One possible way to solve the model-generalization problem is physics-informed DeepONet [30, 27], which combines PINNs and DeepONet. DeepONet was first proposed as a powerful method to learn nonlinear continuous operators [31] from labeled data. It has been further applied to solve parametric PDEs by combining with PINNs [30]. The physics-informed DeepONet method, which differs from the vanilla physics-informed method, can train one neural network that solves parametric PDEs, i.e., the trained neural network can predict the solutions of a class of PDEs given different inputs data. It is physics-informed since, same as the vanilla PINNs, the loss function is composed of constraints of physical laws and labeled data. However, the neural network architecture is similar to the DeepONet method. Accordingly, the inputs are samples of known variables in the domain and additional data that describe the PDE problems. For example, in fluid dynamic problems, the additional data could be PDE coefficients or spatial distribution of source terms. The outputs are still the predicted solutions. Because of this careful design, the neural network is trained simultaneously on a class of PDEs sampled from a high-dimensional space. Once trained, it can predict the solution for an arbitrary set of coefficients for the same type of PDE. Compared with the traditional numerical method, though the training would take a lot more computational resources and time, it is only needed to perform once. Moreover, since the time for prediction on a single unseen PDE is much shorter than the numerical solver, the method is cheaper and efficient when there are

requirements to solve a considerable amount of parametric PDEs. For example, a physics-informed DeepONet model can be trained to solve parametrized Burgers' equations. It was illustrated that the neural network could predict the solution of $\mathcal{O}(10^3)$ equations in a fraction of a second, up to three orders of magnitude faster compared to a conventional PDE solver [30].

The DeepONet method can also improve the performance of PINNs on time-dependent PDEs with large temporal domains [27]. Instead of using the NN as a function approximator of the solution on the whole time interval, Wang et al. [27] trained DeepONets to map random initial conditions to the solutions at time intervals that are much shorter than the whole time domains. Once they train a neural network that can predict the short-time solution on arbitrary initial conditions, to predict the solution at long time intervals, they recursively apply the neural network to evolve the solution forward. Specifically, they first apply the neural network to predict the solution on time interval $(0, \Delta t)$, and then use the solution at Δt, $u(\Delta t)$, as the proposed initial condition to query the neural network for the solution on $(\Delta t, 2\Delta t)$. Repeating the steps above allows them to obtain the solution for the whole time interval. Then they test the method on a series of ODE and PDE examples, including inhomogeneous ODE, stiff chemical kinetics, wave propagation, and diffusion-reaction dynamics. Moreover, they demonstrate that it can get much higher accuracy on long-time integration problems than PINNs, with a fraction of the computational cost compared to classical numerical solvers.

This preliminary work will focus on applying the PINN method to solve problems in short time intervals. In the future, we will explore the physics-informed DeepONet method [30, 27].

1.3 This Work

In this work, we, for the first time, demonstrate that PINNs are able to solve compressible Navier-Stokes equations with exceptionally high accuracy. We identify several issues that prevent the neural networks from converging to global optima:

Initial-condition issue We show that fitting the neural networks to the required initial conditions is challenging and is the main bottleneck for applying the PINN algorithms to our problems.

Output-scaling issue We find that the three outputs of the last fully connected

layers differ by several orders of magnitude, making it difficult for the training to converge.

Gradients-balancing issue We demonstrate that the gradients-balancing problem mentioned previously (Section 1.2) also exists in our problems.

To improve PINNs and to tackle the issues listed above, we:

- apply hard constraints on initial conditions to force the neural networks to satisfy the requirement regardless of the training;
- scale the outputs of the last fully connected layer;
- explore the two methods to balance the contributions of the back-propagated gradients originating from different loss terms to the training: the inverse-initial-loss weighting method and the NTK weighting method [26].

Finally, we test our neural networks against numerical solutions and report the relative $\mathbb{L}_2$ errors (see Equation 4.3).

1.4 Overview

The thesis is structured as follows. In Chapter 2, we discuss the problems we are interested in. In Chapter 3, we explain the collection of testing data. Then we briefly introduce the physics-informed neural networks and discuss several methods to improve its performance on our problems in Chapter 4. Finally, we show the results in Chapter 5 and conclude the thesis in Chapter 6. We include more details in Appendix A.

Chapter 2

The Problem to be Solved

The partial differential equations (PDEs) we would like to solve are

$$\frac{\partial \Sigma}{\partial t} + \nabla \cdot (\Sigma \boldsymbol{v}) = 0 \tag{2.1}$$

$$\frac{\partial (\Sigma v_r)}{\partial t} + \nabla \cdot (\Sigma v_r \vec{v}) - \frac{v_\theta^2}{r} = -\frac{\partial p}{\partial r} - \Sigma \frac{\partial \Phi}{\partial r} + f_r \tag{2.2}$$

$$\frac{\partial (\Sigma r v_\theta)}{\partial t} + \nabla \cdot (\Sigma r v_\theta \vec{v}) = -\frac{\partial p}{\partial \theta} - \Sigma \frac{\partial \Phi}{\partial \theta} + r f_\theta \tag{2.3}$$

The equations are in a two-dimensional (2D) polar coordinate system. It describes a system where gas is moving around a central massive point-source object. In reality, the system evolves in the three-dimensional universe, but since most of the material is located very close a plane, the model is typically simplified to 2D. The gas orbits the central object at near-Keplerian speed. Equation 2.1 is the mass conservation equation. Equation 2.2 and Equation 2.3 are compressible Navier-Stokes equations. The r,θ,t are the spatial-temporal coordinates: r is the radial distance to the center, θ is the angle from a fixed direction, t is the time past since the initial condition of the system. The Σ is vertically-integrated gas surface density and $\boldsymbol{v}$ is velocity vector, while v_r and v_θ are radial and azimuthal velocities.

In this work, as we are looking into the potential of PINNs for this particular PDE for the first time, we focus on the simplest case in protoplanetary disks: there is no planet, and we only consider the interaction of gas and the central star, so the gravitational potential Φ in the equation is the gravitational potential of the central ($r = 0$) massive object. For simplicity, we choose the units to make $G = 1$, $M_\star =$

1, $R_0 = 1$, so

$$\Phi = \frac{1}{r^2} \tag{2.4}$$

The vertically-averaged pressure force (p in the equation 2.2 and 2.3) obey the isothermal assumption in our case, which is determined by solving the equations below

$$p = c_s^2 \Sigma \tag{2.5}$$
$$c_s = h v_K \tag{2.6}$$
$$= h r^{-1/2} \tag{2.7}$$
$$h = h_0 \times r^{FlaringIndex} \tag{2.8}$$

Here c_s is the speed of sound, v_K represents the Keplerian velocity around a central point source, h is the disk aspect ratio and the constant h_0 is the value of h at $r = 1$. $h_0 = 0.05$ and $FlaringIndex = 0.5$ are known constants in our problem.

The last terms in equation 2.2 and 2.3 are projections of viscous force. They are given by

$$\overrightarrow{f} = \nabla \cdot \overrightarrow{\tau} \tag{2.9}$$
$$\overrightarrow{\tau} = 2\eta \overleftrightarrow{D} - \frac{2}{3}\eta(\nabla \cdot \overrightarrow{v}) \overleftrightarrow{I} \tag{2.10}$$
$$\overleftrightarrow{D} = \frac{1}{2}\left[\nabla \overrightarrow{v} + (\nabla \overrightarrow{v})^T\right] \tag{2.11}$$
$$\eta = \Sigma \nu \tag{2.12}$$

here ν is the kinematic viscosity, and we will assume it is constant over the spatial-temporal domain. We will test our PINNs on different values of ν.

Apart from the PDEs, additional information about the initial conditions and boundary conditions is needed to guarantee the uniqueness of the solution. We will study the evolution of a ring-like gas distribution in the disk. The initial condition for Σ and v_r is

$$\Sigma(r, \theta, t = 0) = \Sigma_0 \left(1 + \frac{1}{\sqrt{2\pi}\Delta} e^{-\frac{(r-r_c)^2}{2\Delta^2}}\right) \tag{2.13}$$
$$v_\theta(r, \theta, t = 0) = r^{-1/2} \tag{2.14}$$

r_c denotes the center of the ring while Δ denotes the initial ring width. The initial

Case	kinematic viscosities	time (orbit)
I	0.01	0.04
II	0.001	0.2
III	0.0	0.2

Table 2.1: Setups of three PDE cases. The cases have common setups except kinematic viscosites and temporal ranges of the PDE.

condition of v_θ is given by the equation

$$rv_r\Sigma = -3\nu r^{1/2}\frac{\partial}{\partial r}\left(r^{1/2}\Sigma\right) \tag{2.15}$$

See the left column of Figure 3.1 for the visualization of the initial conditions. All unknown variables have periodic boundary conditions in the azimuthal direction. The radial boundaries are at $r_{min} = 0.4$ and $r_{max} = 2.5$. The requirements of Σ, v_r and v_θ on radial boundaries are given by

$$\frac{\partial\Sigma}{\partial r}(r = r_{max}\ or\ r_{min}, \theta, t) = 0 \tag{2.16}$$

$$\frac{\partial v_r}{\partial r}(r = r_{max}\ or\ r_{min}, \theta, t) = 0 \tag{2.17}$$

$$\frac{\partial v_\theta}{\partial r}(r = r_{max}\ or\ r_{min}, \theta, t) = -\frac{v_\theta}{2r} \tag{2.18}$$

We will explore the performance of PINNs on the disk evolution with different viscosities. We choose the kinematic viscosities to be 10^{-2} (high viscosity), 10^{-3} (low viscosity) and 0 (no viscosity).

The time domain of the problem (Table 2.1) is determined by the simulation, and we limit the time domain so that the diffusion wave does not reach the inner radial boundary to avoid complex situations where the reflection from the inner boundary has a significant influence on the material inside the domain.

Chapter 3

Hydrodynamic Simulations

We use the hydrodynamic simulation code FARGO3D [1],[2] to obtain numerical solutions of the PDEs described in chapter 2. FARGO3D solves the PDEs using conventional techniques (finite-difference upwind, dimensionally split methods). The numerical solutions will be used as ground truth to test the accuracy of our NN. We show the solution of Σ, v_r and v_θ at time steps of 0.0 orbits, 0.1 orbits and 0.2 orbits for PDE with $\nu = 0.001$ in Figure 3.1.

Since the PDEs and their solutions are axisymmetric, we only plot the radial profiles of unknown variables for the $\nu = 0.0$ (Figure 3.3) and $\nu = 0.01$ (Figure 3.2) cases. For the same reason, when comparing the predictions from our neural networks with the numerical solutions displayed here, we also only plot the radial profiles (see Figure 5.1).

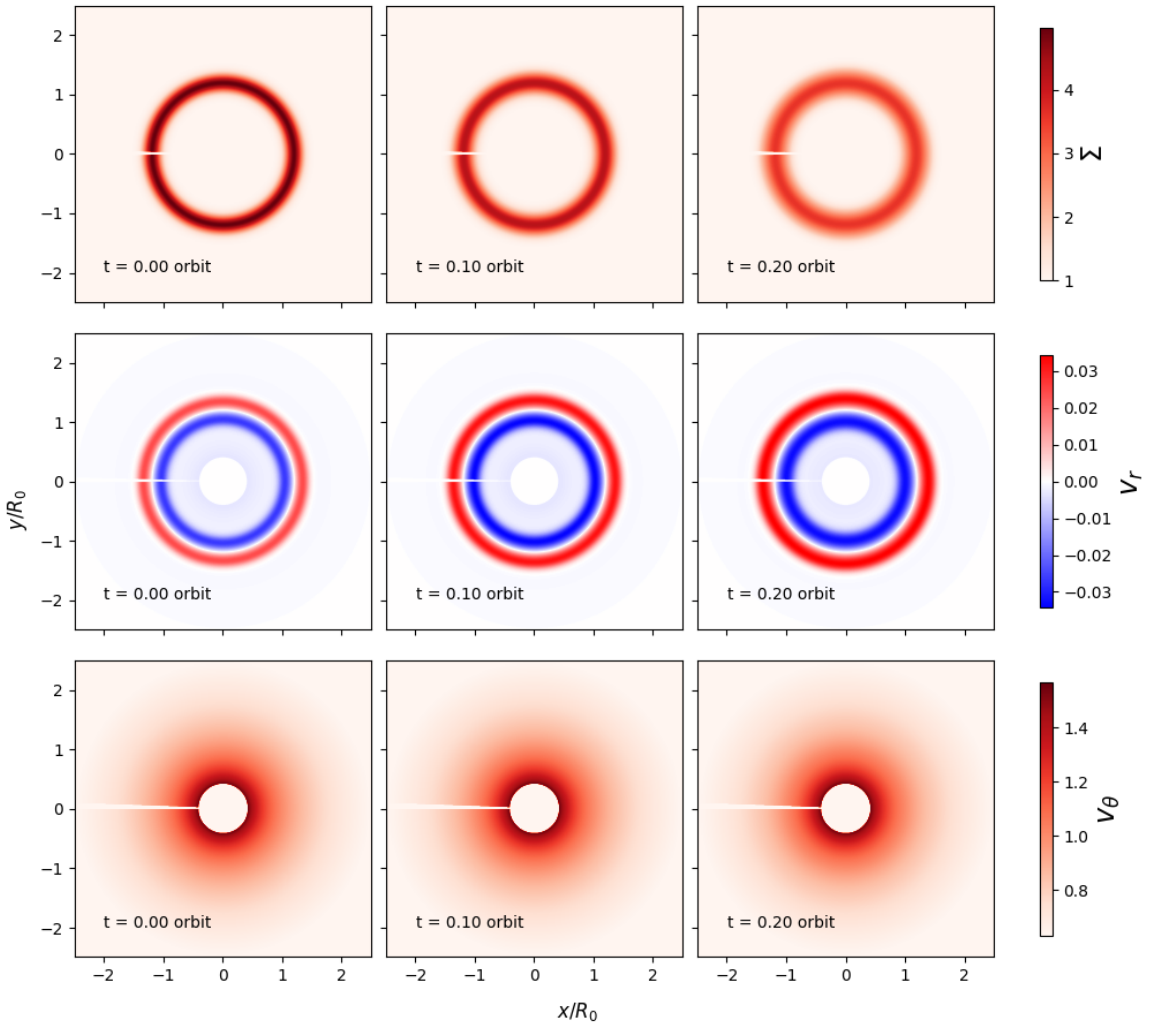
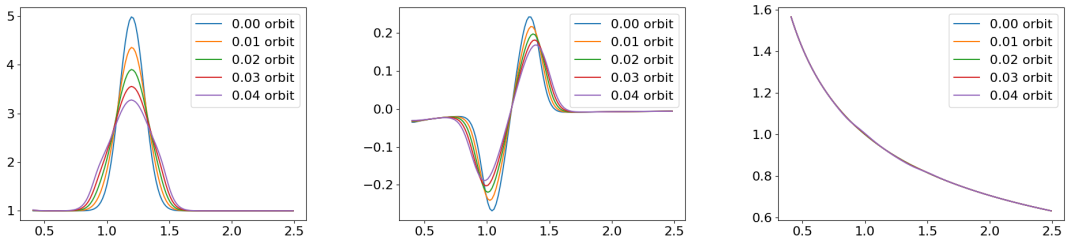
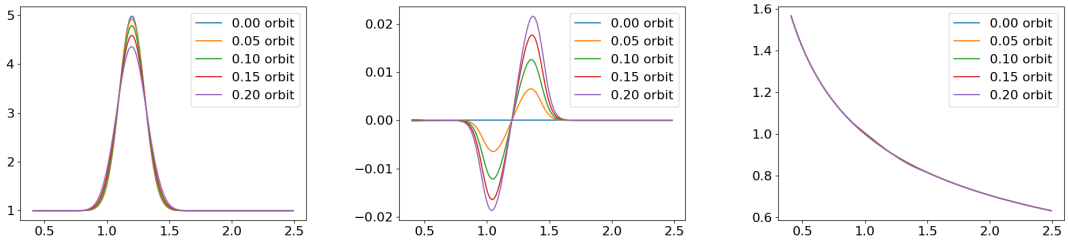


Figure 3.1: Snapshots of 2D numerical solutions of the PDEs with a kinematic viscosity $\nu = 0.001$ (case II in Table 2.1), generated using the hydrodynamic simulation program, FARGO3D. Top: Gas density distribution (Σ). Middle: Radial velocity distribution (v_r). Bottom: Azimuthal velocity distribution (v_θ).



(a) Gas density (Σ) (b) Radial velocity (v_r) (c) Azimuthal velocity (v_θ)

Figure 3.2: The radial profiles of numerical solutions for PDEs with $\nu = 0.01$ (case I in Table 2.1). The data is generated by FARGO3D [1, 2]. The solutions are used as ground truth to compare with our PINNs' predictions.



(a) Gas density (Σ) (b) Radial velocity (v_r) (c) Azimuthal velocity (v_θ)

Figure 3.3: The radial profiles of numerical solutions for PDEs with $\nu = 0.0$ (case III in Table 2.1). The data is generated by FARGO3D [1, 2]. The solutions are used as ground truth to compare with our PINNs' predictions.

Chapter 4

Method

4.1 Fundamental: How do the Vanilla PINNs Work

As described in Chapter 1, we will explore the application of physics-informed neural networks [14] on solving compressible Navier-Stokes equations (see Chapter 2) in protoplanetary disks. Specifically, we will focus on the forward problem: given a set of PDEs with varying but known coefficients, we ask PINNs to predict the solutions of the equations.

We follow the neural network architecture of [14] with the modifications described in the following subsections (see Figure 4.1 for the illustration of the architecture). The neural network, as a universal function approximator [32], approximates the solution function. The physics-informed loss function, composed of PDE residuals and boundary/initial condition mean squared errors, penalizes the neural network to bias toward the correct solutions.

4.1.1 Neural Networks

Figure 4.1 shows the neural network used in this work. The inputs to the neural network are samples of coordinates, which in our problems are radial coordinate (r), azimuthal coordinate (θ) and time (t). Different from the Convolutional Neural Networks (CNN), the inputs are not necessarily limited to regular grids. Rather, arbitrary positions inside the domain are acceptable. The inputs are first transformed by the inputs transform layer[1] and then passed through fully connected hidden layers. Unless specified explicitly, the neural networks we use have hidden layers of the size

[1]For normalization of the inputs and periodic boundary condition (see A.1).

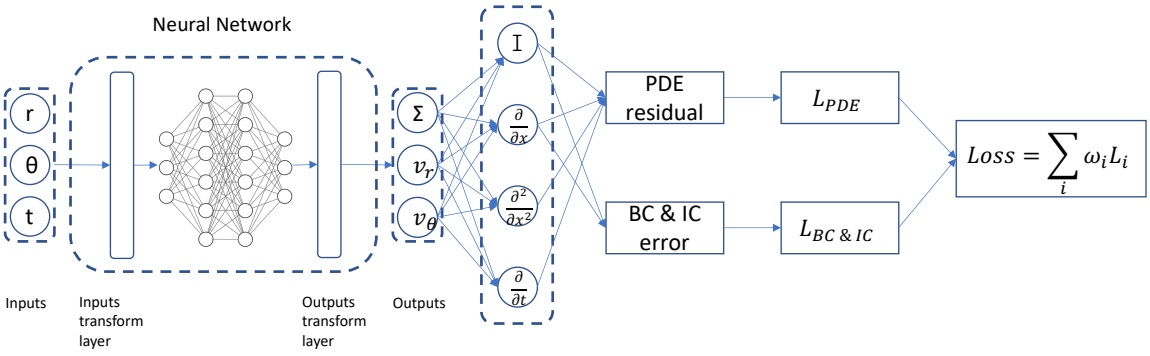


Figure 4.1: The architecture of the physics-informed neural networks. The inputs are radial coordinate (r), azimuthal coordinate (θ) and time (t). The inputs are first sent to the input transform layer to enforce hard constraints on periodic boundary conditions, then passed through the fully connected hidden layers. The outputs of the last fully connected layer are sent to the outputs transform layer (see Section 4.3.1 and Section 4.3.2) to obtain the final outputs. The final outputs represent the predicted solutions of Σ, v_r and v_θ. To evaluate the losses, we apply several mathematical operators to the final outputs: the I represents the identity mapping, the $\frac{\partial}{\partial x}$ and the $\frac{\partial^2}{\partial x^2}$ represent the first-order and second-order differential operators of the spatial coordinates respectively, and the $\frac{\partial}{\partial t}$ represents the first-order differential operator of the time. The outputs and the partial derivatives of the outputs compose the PDE residual, BC errors and IC errors, which compose the loss functions.

32×9. The outputs of the hidden layers are passed to a fully connected layer with three neurons[2]. Then the outputs of the last layer are transformed by another transform layer (see Section 4.3.1 and Section 4.3.2) to generate the final outputs of the neural network. For any inputs (r_i, θ_i, t_i), the outputs represent the predicted solutions at (r_i, θ_i, t_i): $\Sigma(r_i, \theta_i, t_i)$, $v_r(r_i, \theta_i, t_i)$ and $v_\theta(r_i, \theta_i, t_i)$.

4.1.2 Loss Functions

Instead of using labeled data from experiments or observations to train the network, the physics-informed method uses physical laws to train the network. Accordingly, the loss function is composed of three parts: PDE losses, boundary condition (BC) losses and initial condition (IC) losses.

The PDE losses (The L_{PDE} in Figure 4.1) measure to what extent the neural network's prediction obeys the physical laws. They are computed by the mean squares of the PDE residual. For example, if the equation to be solved is

$$\mathcal{L}u(x) = 0 \tag{4.1}$$

Where $\mathcal{L}$ is an arbitrary nonlinear differential operator. Then the PDE loss is given by

$$L_{PDE} = \frac{\sum_i^N \left[\mathcal{L}u(x_i)\right]^2}{N} \tag{4.2}$$

The index is summed over a batch of N input coordinates. For our problems, since we have three equations, there are three PDE loss terms in the loss function.

Similarly, the BC losses and IC losses (The $L_{BC\&IC}$ in Figure 4.1) are computed by the mean squared errors of boundary conditions and initial conditions. The spatial domain in our problem is $r \in [0.4, 2.5]$ and $\theta \in [-\pi, \pi]$, so the boundaries include $r = 0.4$, $r = 2.5$, $\theta = -\pi$ and $\theta = \pi$. Our periodic boundary conditions are enforced by hard constraints (see Appendix A.1). As a result, we only include BC losses at the inner and outer radial boundaries. At each boundary, there are three loss terms for the three unknown variables (Σ, v_r and V_θ). Therefore, we have six BC loss terms in total. In addition, we have three IC loss terms for the initial conditions of the three variables.

All these terms described above compose the total loss function, which we would like to minimize. Like traditional deep learning training, we minimize the loss function

[2]The layer does not have activation function.

by taking gradients with respect to the neural network parameters and then updating the parameters based on the gradient descent direction. We use Adam optimizer [33]. Once the loss function is minimized, the neural network obeys the PDEs, and the neural network is a close approximation of the correct solution.

In the calculation of PDE losses and BC losses, partial derivatives of unknown variables with respect to spatial and temporal coordinates are required. They are calculated using the same automatic differentiation technique as back-propagation, which has already been supported by popular deep learning frameworks such as TensorFlow [34], PyTorch [35] and JAX [36]. To make sure the neural network function is second-order differentiable, we use the sine activation function in hidden layers.

4.1.3 Training and Testing Process

In the training process, we randomly sample a batch of coordinates from the spatial-temporal domain, send to the neural network, and obtain the predicted solutions at these coordinates. We also obtain the partial derivatives at the same coordinates and compute the PDE and BC/IC loss terms. Then we compute the gradients and update the neural network. We repeat the steps above with sufficient iterations to approach an optimized neural network.

In the testing process, we introduce the data from hydrodynamic simulations in Chapter 3. We send the same coordinates in the simulations to the trained neural network and compare the outputs with numerical solutions. We calculate the relative $\mathbb{L}_2$ errors for all the unknown variables:

$$\text{relative } \mathbb{L}_2 \text{ error} = \frac{\left[\sum_i^N (u_{predict} - u_{truth})^2\right]^{1/2}}{\left(\sum_i^N u_{truth}^2\right)^{1/2}} \tag{4.3}$$

The index i is summed over a batch of N input coordinates. The $u_{predict}$ represents the prediction of the neural networks. The u_{truth} represents the ground truth.

4.2 When the Vanilla PINN Fails

We first try the vanilla PINN, given by a fully connected neural network and a loss function of

$$L = \sum_i L_{IC,i} + \sum_j L_{BC,j} + \sum_k L_{PDE,k} \tag{4.4}$$

The right-hand side of the Equation 4.4 is the sum of loss terms that measure the fitting of the neural network on initial conditions, radial boundary conditions, and PDEs. The index i is summed over all the ICs in the loss function. The index j is summed over all the BCs in the loss function. The index k is summed over all the PDEs in the loss function. The periodic boundary conditions are enforced by adding an input transform layer between the inputs and the hidden layers in Figure 4.1 (see Appendix A.1).

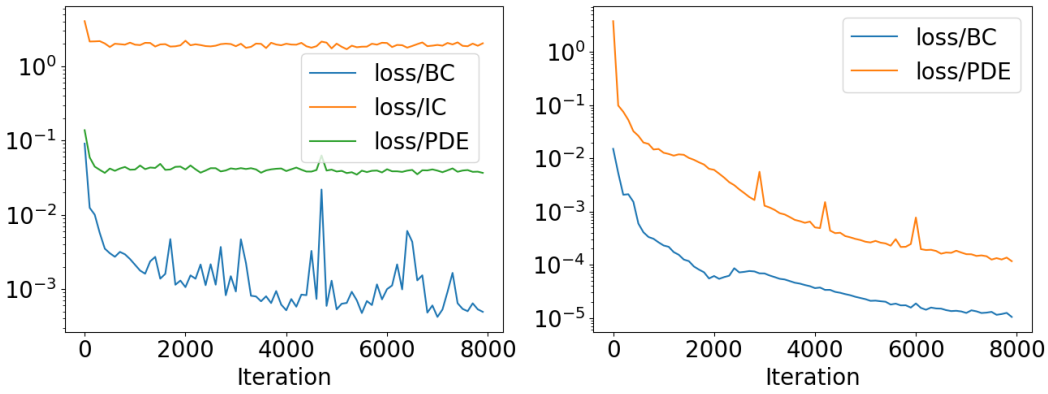
For case II in Table 2.1, we train the neural network with hyper-parameters in Appendix A.2.1 and the neural network does not converge (see the relative $\mathbb{L}_2$ errors in the first row of Table 4.1).

To explore the reason, we plot the loss curves in Figure 4.2a. Note that the loss terms, as indicated in Section 4.1.2, are composed of 12 terms: six terms for the boundary conditions of three unknown variables at $r = r_{min}$ and $r = r_{max}$, three terms for the initial conditions of three unknown variables, and three terms for three partial differential equations. For simplicity, we sum the terms of the same type and only plot the total boundary condition losses, total initial condition losses and total PDE losses in Figure 4.2a and later loss plots.

In Figure 4.2a, the sum of IC losses is more than one order of magnitude larger than the sum of BC losses and more than three orders of magnitude larger than the sum of PDE losses. Furthermore, the sum of IC losses does not decrease after the initial several hundred steps (Figure 4.2a). We infer that

- It is hard to train the neural network to fit the initial conditions for our problems.
- Since the magnitude of gradients is proportional to the magnitude of loss terms, the gradients are easily be dominated by the ICs, which prevents the neural networks from being optimized to fit the PDE and BCs loss terms.

We also observe that our vanilla neural network has the same IC fitting problem with highly similar loss curves for PDE cases I and III (Figure A.1a, A.2a).



(a) (b)

Figure 4.2: Loss curves for PDE problem case II ($\nu = 0.001$), obtained from two neural networks. There are no weights to balance the gradients originating from different loss terms and no output scaling in either. (a) A vanilla PINN. Blue: total boundary condition losses. Orange: total initial condition losses. Green: total PDE losses. (b) A PINN with hard constraints on initial conditions. Blue: total boundary conditions losses. Orange: total PDE losses.

4.3 Improvements on the Vanilla PINN

4.3.1 Architecturally Enforcing Initial Conditions

In Section 4.2, we explore constraining the neural networks by including the errors on ICs in loss functions. We find that the ICs are hard to fit and dominate the gradients, thus preventing the neural networks from converging to satisfy PDEs and BCs. So we purse to use hard constraints to enforce the requirements of initial conditions without training [28]. Specifically, we add another transformation layer after the last layer of the neural network. The general idea is to suppress the activity of the neural networks on inputs at $t = t_0$ and replace the network outputs (predictions) with the ICs. So we have the transformation

$$u(r, \theta, t) = u_{ic}(r, \theta) + A(t) \times \hat{u}(r, \theta, t) \tag{4.5}$$

Here u represents Σ, v_r and v_θ, and u_{ic} represents the initial conditions, which are known in our problems. $\hat{u}(r, \theta, t)$ is the direct output of the last fully connected layer. The aim of the function $A(t)$ here is to suppress the activity of the neural network at $t = t_0$, so we choose the form

$$A(t) = 1.0 - Exp\left(-\frac{t - t_0}{t_{max} - t_0}\right) \tag{4.6}$$

Since $A(t = t_0) = 0$, after the transformation we have $u(r, \theta, t = t_0) \equiv u_{ic}(r, \theta)$ for any neural network parameters, which means initial conditions are satisfied without training. Therefore, we do not need to include constraints of initial conditions in losses.

In a new experiment, We train the neural network with the same hyper-parameters as in Section 4.2 for the three PDE cases, but we apply hard constraints on initial conditions. The loss curves for PDE case II are shown in Figure 4.2b, which show that the BC and PDE losses decrease smoothly, and the final losses of BCs and PDEs are one and two orders of magnitude lower than those in the vanilla PINN. The relative $\mathbb{L}_2$ errors for case II are in Table 4.1, which indicate that we achieve a much better fit than that in the neural network without hard constraints on initial conditions (Σ: errors reduced by 97%, v_r: errors reduced by 99.5%, v_θ: errors reduced by 99.4%). The results of PDE cases I (Figure A.1, Table A.2) and III (Figure A.2, Table A.3) also show similar phenomenons and support our conclusion above. As a

Relative errors	Σ	v_r	v_θ
Soft constraints on ICs	0.821	61.3	0.309
Hard constraints on ICs	**0.0237**	**0.288**	**0.001 82**

Table 4.1: Relative $\mathbb{L}_2$ errors (Equation 4.3), compared to the ground truth, in neural networks with soft and hard constraints on initial conditions. The neural networks are trained on PDE case II ($\nu = 0.001$). The errors are averaged over three repetitive runs. Soft constraint: The initial conditions are constrained by minimizing the corresponding loss terms. Hard constraint: The initial conditions are enforced by an output transformation layer (see Equation 4.5).

result, removing the IC terms in loss functions can significantly reduce the imbalance among loss terms and thus ease the difficulty in training.

4.3.2 Output Scaling

We further improve the fitting by applying output scaling. After applying hard constraints on ICs to our neural network (Section 4.3.1), the expected outputs of the last fully connected layer changed from u to $\hat{u}$ (see Equation 4.5). From the ground truth of PDE case II ($\nu = 0.001$) we can see that the expected magnitude of $\hat{\Sigma}$ is around 1, the expected magnitude of $\hat{v_r}$ is around 10^{-2}, and the expected magnitude of $\hat{v_\theta}$ is around 5×10^{-3}. Therefore, there are large differences among the magnitudes of the outputs of the last fully connected layer. Previous works have shown that properly scaling these outputs may significantly improve the predictive accuracy [27, 19]. So we explore applying output scaling between the last fully connected layer and the outputs transform layer.

The next problem is how to determine the scaling factors. Since we should not introduce outside information about the ground truth in training, the best choice is to make initial guesses of the scaling factors and perform hyper-parameter tuning to get the optimal values. We fix the scaling factor for $\hat{\Sigma}$ to 1 and sample the scaling factors for $\hat{v_r}$ and $\hat{v_\theta}$ uniformly on log scales. Then we test our trained neural networks on ground truth to obtain the optimal scaling factors.

Applying output scaling improves our neural networks' predictive accuracy. The optimal scaling factors for case II ($\nu = 0.001$) are 0.01 for v_r and 0.001 for v_θ. Applying output scaling leads to a significant improvement compared with the neural network only applying hard constraints on ICs (Σ: errors reduced by 52%, v_r: errors reduced by 61%, v_θ: errors reduced by 65%) (Table 4.2). We find a similar result for case III ($\nu = 0.0$): applying output scaling can significantly reduce the relative $\mathbb{L}_2$ errors for all the unknown variables (Σ: errors reduced by 76%, v_r: errors reduced by 76%, v_θ: errors reduced by 49%) (Table A.4). However, output scaling does not significantly improve prediction accuracy for case I ($\nu = 0.01$). The reason may be that the differences in magnitudes among the $\hat{u}$ are not significant, thus the training does not suffer as much on the scaling.

Relative errors	Σ	v_r	v_θ
Without output scaling	0.0237	0.288	0.001 82
With optimal output scaling	**0.0113**	**0.111**	**0.000 635**

Table 4.2: Relative $\mathbb{L}_2$ errors (Equation 4.3), compared to the ground truth, of neural networks without output scaling and with optimal scaling factors for PDE case II ($\nu = 0.001$), averaged over three repetitive runs. Both neural networks are hard constrained on initial conditions, trained for 8000 steps.

4.3.3 Gradients Balancing

In this section, we study the contributions of the back-propagated gradients originating from different terms in PINNs loss functions to the training and balance them using the two state-of-the-art methods.

We estimate the magnitudes of gradients originating from PDE and BC losses for the best neural network from the previous section (see Figure 4.3a). The gradients are estimated by averaging the absolute values among inputs, neurons, layers, and loss terms. We note that the selection of the methods for estimating the magnitudes of gradients should not have significant influences on our results.

From Figure 4.3a, the PDE gradients are around two orders of magnitude larger than BC gradients, which indicates that the requirement of fitting the PDEs dominates the training process in the neural network without gradients balancing. Moreover, the PDE loss terms decrease gradually while BC loss terms almost do not decrease after 2000 steps (see Figure 4.3b). This phenomenon also indicates that the training is dominated by the fitting to PDEs. The issue prevents the neural networks from converging to global optima.

To fix the issue, we apply weights to balance the gradients. In this work, we propose using the initial values of loss terms to balance the gradients (Inverse-initial-loss weights). We also explore the application of the Neural Tangent Kernel (NTK) adaptive weighting method [26] to our problems.

Inverse-initial-loss weights The losses at the 0^{th} training step are easy to obtain (evaluated based on the initial guesses). Moreover, the initial losses might indicate the relative magnitudes of loss terms and the relative magnitudes of gradients. So we propose to weight the gradients by the initial losses. Without the weights, the total gradient is

$$g_{total} = \sum_i g_i \tag{4.7}$$

The index is summed over the gradients originating from different loss terms. Then after applying the technique, the new total gradient is

$$g_{total} = \sum_i w_i g_i \tag{4.8}$$

where $w_i = \frac{1}{L_{i,init}}$ are the inverses of the loss terms at the beginning of training, i.e., the inverse of the loss terms evaluated on untrained neural networks.

Relative errors	Σ	v_r	v_θ
No weights	0.0113	0.111	**0.000 635**
Inverse-initial-loss weights	0.009 07	0.0718	0.001 23
NTK weights	**0.008 95**	**0.0684**	0.001 19

Table 4.3: Relative $\mathbb{L}_2$ errors (Equation 4.3), compared to ground truth, of neural networks with different gradients-balancing strategies on loss terms: No weighting, inverse-initial-loss weighting and NTK weighting. The errors are averaged over three repetitive runs. The neural networks are trained for PDE case II ($\nu = 0.001$) with 8000 steps. All neural networks have hard constraints on initial conditions and optimal output scaling.

Neural Tangent Kernel (NTK) weights Another way to balance the gradients is the NTK weighting method [26]. The method assigns weights to gradients by estimating the convergence rate of different loss terms, which can further be calculated by the trace of the Neural Tangent Kernel matrices.

For either of the two methods above, applying weights to the gradients can balance the gradients originating from different loss terms, thus making the updates of the neural networks reflect the requirements of fitting all loss terms. We explore both methods and compare the outcomes with that from the neural network without applying weights. For case II ($\nu = 0.001$, Table 4.3), the two gradients-balancing methods generate equally good outcomes, with the relative $\mathbb{L}_2$ errors of Σ reduced by 20% and of v_r reduced by 35%, much better than the neural network without weighting. However, the fitting on v_θ is worse than that of the neural network without weighting.

To demonstrate the effect of the gradient balancing methods, we compare the training loss curves of neural networks with and without gradient balancing in Figure 4.3. For the neural network without gradients balancing (Figure 4.3b), because the PDE gradients are dominating the training, the BC losses do not decrease much after 2000 steps. However, for the neural network with NTK weighting (Figure 4.3c), the BC loss terms decrease at the same pace as the PDE loss terms.

In PDE case III ($\nu = 0.0$, Table A.5), we get similar results as in case II above: applying either adaptive weighting methods above yield significant improvements. Without the training being dominated by the PDEs, the BC losses and PDE losses can decrease simultaneously (Figure A.3). We also compare the outcomes of the two methods: the Inverse-initial-loss weighting method produces smaller relative $\mathbb{L}_2$

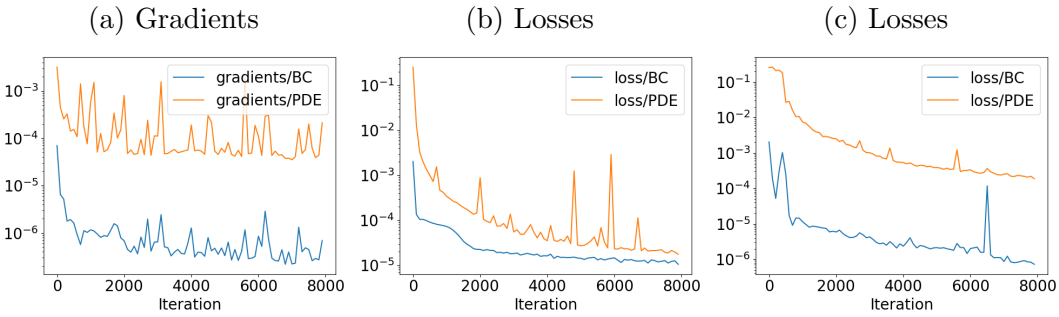


Figure 4.3: (a) Magnitudes of gradients with training iterations for PDE problem case II ($\nu = 0.001$). The gradients are estimated by averaging the absolute values among inputs, neurons, layers and loss terms. The PDE gradients is two orders of magnitude larger than the BC gradients, indicating that the requirement of fitting PDEs dominates the training if there are no weights. (b) Loss curves for PDE problem case II ($\nu = 0.001$). The neural network has hard constraints on ICs and optimal output scaling, but does not have weights for gradients. The BC losses almost do not decrease after 2000 iterations. (c) Loss curves for PDE problem case II ($\nu = 0.001$). The neural network has hard constraints on ICs, optimal output scaling and NTK weighting method. The BC losses and PDE losses decrease simultaneously.

errors of Σ and v_r but a bigger error of v_θ than the NTK weighting method (see Table A.5).

Chapter 5

Results

We test the methods discussed above (see Chapter 4) in the three PDE cases that we are interested in (Table 2.1). We find that for the cases with low viscosity ($\nu = 0.0001$) and no viscosity, applying the methods below at the same time will produce the best results

- Hard constraints on initial conditions
- Scaling on neural network outputs
- Balancing the back-propagated gradients originating from different terms in PINNs loss functions

Missing any of them would cause the final errors to increase by $20\% \sim 1000\%$. Similarly, applying hard constraints on the initial conditions improves the outcomes significantly in the high viscosity case ($\nu = 0.01$), However, applying output scaling and (or) adaptive weights on loss terms do not significantly improve the fitting. This may partially be explained as that the differences in magnitudes of the expected outputs of the last fully connected layer among the unknown variables in the $\nu = 0.01$ case are the smallest. Thus it might not suffer as much from the output scaling issue.

The prediction, compared with the ground truth, is shown in Figure 5.1. We also report the relative errors in Table 4.3. Note that the solutions for our problems are 2D. However, since they should be azimuthal symmetric, we only plot the azimuthally-averaged values.

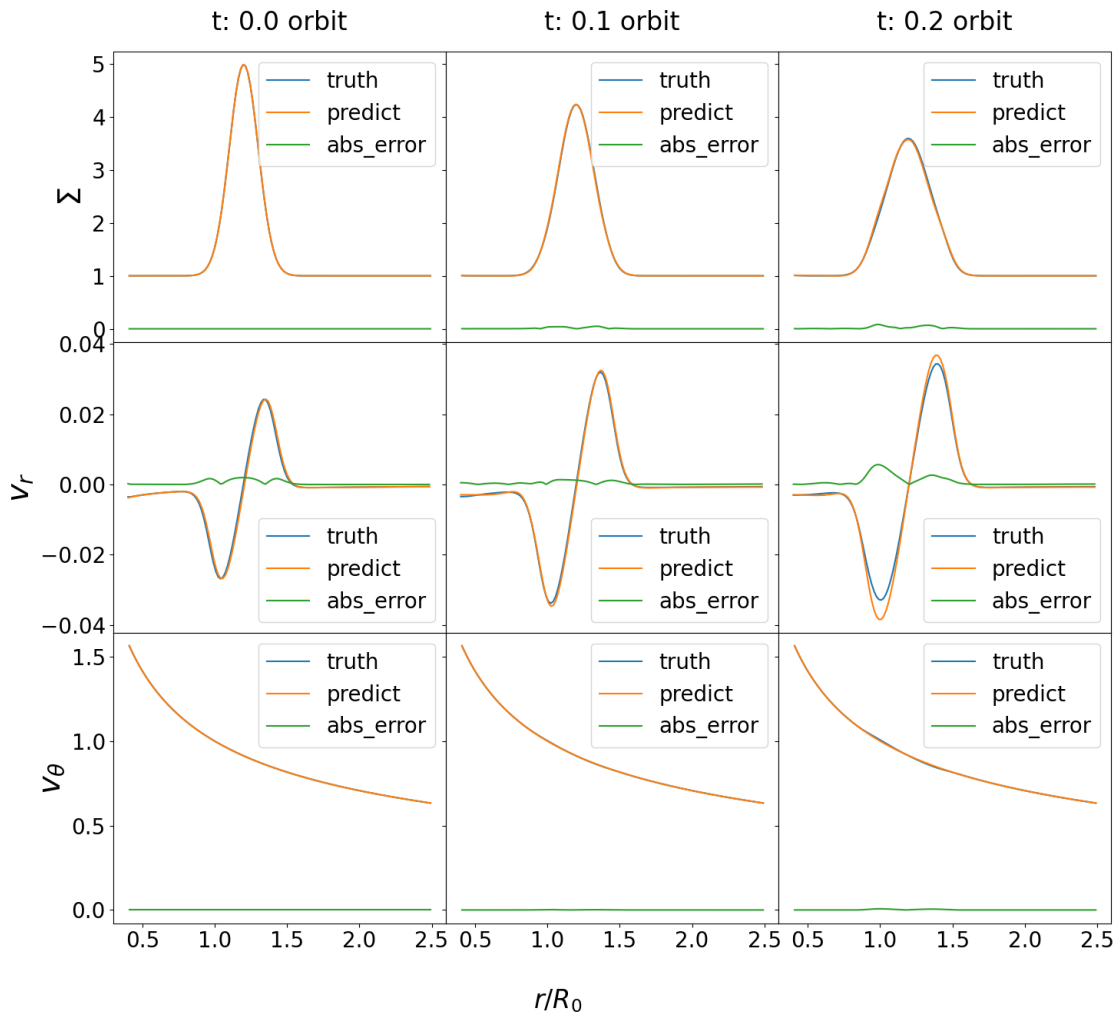


Figure 5.1: The 1D radial profiles of the PINN's solution (orange), the ground truth (blue) and the absolute error (green) of the best neural network for PDE case II ($\nu = 0.001$). The ground truth is obtained from a FARGO3D simulation.

Chapter 6

Conclusion

For the first time, we explore the application of physics-informed neural networks in solving compressible Navier-Stokes equations in protoplanetary disks. We identify several issues below that hinder the applications of vanilla PINNs:

Initial-condition fitting issue When the initial-condition constraints are part of the loss functions, it is challenging to train the neural networks and they often fail to converge.

Output-scaling issue The three outputs of the last fully connected layers differ by several orders of magnitude.

Gradients-balancing issue The back-propagated gradients are not balanced in the training process. The loss functions of PINNs are usually composed of several terms: residuals of PDEs, errors on initial conditions, errors on boundary conditions, and errors on observed data. The gradients with respect to the NN parameters, derived from these loss terms respectively, may numerically differ by several orders of magnitude [25], [26]. Therefore, if we simply sum these gradients up to yield the direction of updating the NN parameters, a few large terms would dominate the training and prevent the neural networks from converging to global optima.

We explore several techniques to tackle the issues above:

- Applying hard constraints on initial conditions to force the neural networks to satisfy the requirements regardless of the training.
- Scaling the outputs of the last fully connected layer manually.

- Exploring two methods to balance the contributions of back-propagated gradients originating from different loss terms: Inverse-initial-loss weights and NTK weights [26].

We test our neural network on ground truth solutions generated using hydrodynamic simulation software (FARGO3D [2] and calculate relative $\mathbb{L}_2$ errors. With the techniques listed above, we reduce the errors by $97\% \sim 99\%$ for the three PDE cases compared to the vanilla PINNs. We report the relative $\mathbb{L}_2$ errors for the three PDEs we tested: for PDE case I ($\nu = 0.01$), the errors are 1.6% for gas density, 7.0% for radial velocity and 0.090% for azimuthal velocity (Table A.2); for PDE case II ($\nu = 0.001$), the errors are 0.90% for gas density, 6.8% for radial velocity and 0.12% for azimuthal velocity (Table 4.3); for PDE case III ($\nu = 0.0$), the errors are 0.16% for gas density, 9.5% for radial velocity and 0.086% for azimuthal velocity (Table A.5).

Appendix A

Additional Information

A.1 Periodic Boundary Condition

We enforce hard constraints on periodic boundary condition by input transformation. The requirement of periodic boundary condition is that $u(r, \theta = 0, t) = u(r, \theta = 2\pi, t)$ for u as Σ, v_r and v_θ. This can be enforced without training by applying a transformation between the inputs and the first hidden layer

$$(r, \theta, t) \rightarrow (r, sin\theta, cos\theta, t) \tag{A.1}$$

Since the value and any order of derivatives is equal for $sin\theta$ and $cos\theta$ for $\theta = 0$ and $\theta = 2\pi$, the output of the neural network will obey $u(r, \theta = 0, t) = u(r, \theta = 2\pi, t)$ for u as Σ, v_r and v_θ, thus satisfies the periodic boundary condition.

A.2 Hyper-parameters

A.2.1 Neural Network in Section 4.2

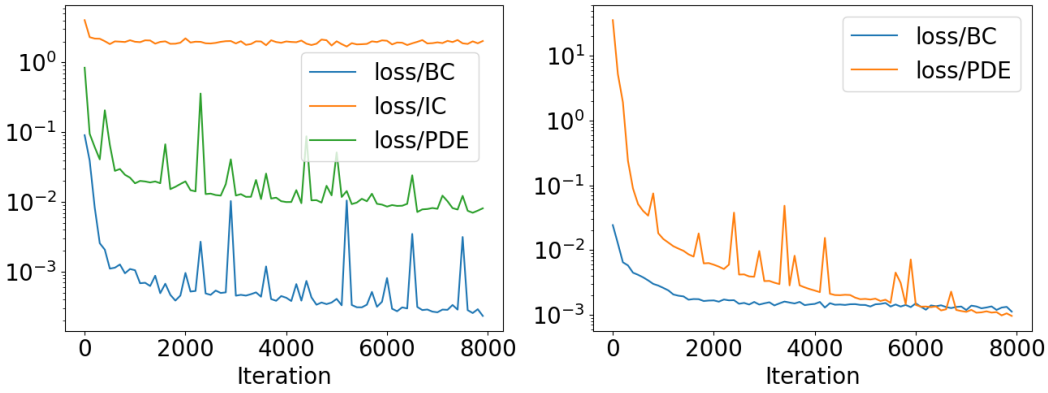
Here we show the main hyper-parameters used for the training in section 4.2 (Table A.1).

A.3 Loss Curves

A.4 Relative Errors

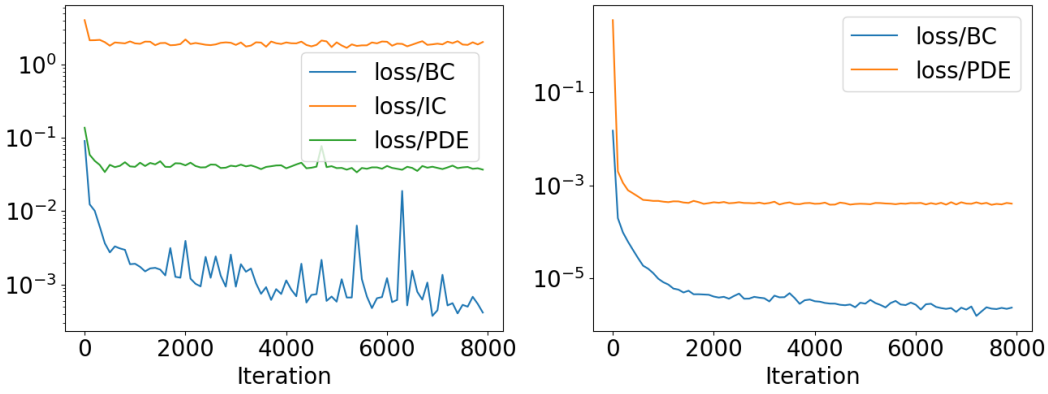
Hyper-parameter names	**Value**
Activation function	sin
Optimizer	Adam
Learning rate	0.001
Learning rate exponential decay	0.9
Learning rate decay transition steps	1000
Hard constraints on ICs	False
Loss weights	Null
Initializer	glorot normal
Hidden layer size	32×9
Batch size on boundary	1000
Batch size at initial	1000
Batch size inside domain	4000
Number of steps	4000
Scaling factor for Σ	1.0
Scaling factor for v_r	1.0
Scaling factor for v_θ	1.0

Table A.1: Hyper-parameters for the training at section 4.2.



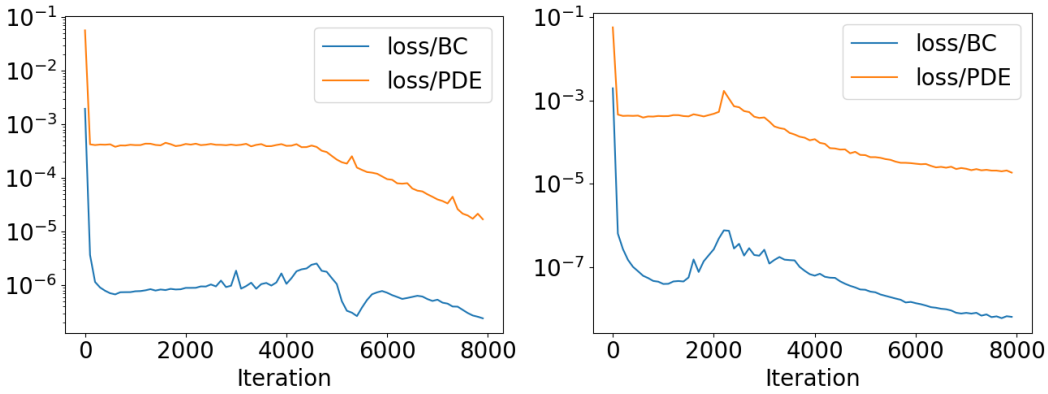
(a) (b)

Figure A.1: Loss curves for PDE problem case I ($\nu = 0.01$), obtained from two neural networks. There are no weights to balance the gradients originating from different loss terms and no output scaling in either. (a) A vanilla PINN. Blue: total boundary condition losses. Orange: total initial condition losses. Green: total PDE losses. (b) A PINN with hard constraints on initial conditions. Blue: total boundary conditions losses. Orange: total PDE losses.



(a) (b)

Figure A.2: Loss curves for PDE problem case III ($\nu = 0.0$), obtained from two neural networks. There are no weights to balance the gradients originating from different loss terms and no output scaling in either. (a) A vanilla PINN. Blue: total boundary condition losses. Orange: total initial condition losses. Green: total PDE losses. (b) A PINN with hard constraints on initial conditions. Blue: total boundary conditions losses. Orange: total PDE losses.



(a) No weights. (b) Inverse-initial-loss weights.

Figure A.3: Loss curves for PDE problem case III ($\nu = 0.0$), both with hard constraints on ICs and optimal output scaling. Left: A PINN without gradient weighting. PDE gradients dominate the training, the BC losses do not decrease. Right: A PINN with NTK weighting method. The gradients are balanced and the loss terms decrease simultaneously. Blue: total boundary condition losses. Orange: total PDE losses.

Relative errors	Σ	v_r	v_θ
Soft constraints on ICs	0.680	9.73	0.384
Hard constraints on ICs	**0.0161**	**0.0699**	**0.000 905**

Table A.2: Relative $\mathbb{L}_2$ errors (Equation 4.3), compared to the ground truth, in neural networks with soft and hard constraints on initial conditions. The neural networks are trained on PDE case I ($\nu = 0.01$). The errors are averaged over three repetitive runs. Soft constraint: The initial conditions are constrained by minimizing the corresponding loss terms. Hard constraint: The initial conditions are enforced by an output transformation layer (see Equation 4.5).

Relative errors	Σ	v_r	v_θ
Soft constraints on ICs	0.830	156	0.310
Hard constraints on ICs	**0.0455**	**0.987**	**0.001 44**

Table A.3: Relative $\mathbb{L}_2$ errors (Equation 4.3), compared to the ground truth, in neural networks with soft and hard constraints on initial conditions. The neural networks are trained on PDE case III ($\nu = 0.0$). The errors are averaged over three repetitive runs. Soft constraint: The initial conditions are constrained by minimizing the corresponding loss terms. Hard constraint: The initial conditions are enforced by an output transformation layer (see Equation 4.5).

Relative errors	Σ	v_r	v_θ
Without output scaling	0.0455	0.987	0.001 44
With optimal output scaling	**0.0110**	**0.241**	**0.000 734**

Table A.4: Relative $\mathbb{L}_2$ errors (Equation 4.3), compared to the ground truth, of neural networks without output scaling and with optimal scaling factors for PDE case III ($\nu = 0.0$), averaged over three repetitive runs. Both neural networks are hard constrained on initial conditions, trained for 8000 steps.

Relative errors	Σ	v_r	v_θ
No weights	0.0110	0.241	0.000 734
Inverse-initial-loss weights	**0.001 65**	**0.0951**	0.000 864
NTK weights	0.004 60	0.0952	**0.000 415**

Table A.5: Relative $\mathbb{L}_2$ errors (Equation 4.3), compared to ground truth, of neural networks with different gradients-balancing strategies on loss terms: No weighting, inverse-initial-loss weighting and NTK weighting. The errors are averaged over three repetitive runs. The neural networks are trained for PDE case III ($\nu = 0.0$) with 8000 steps. All neural networks have hard constraints on initial conditions and optimal output scaling.

Bibliography

[1] F. Masset. FARGO: A fast eulerian transport algorithm for differentially rotating disks. *Astronomy and Astrophysics Supplement Series*, 141:165–173, January 2000.

[2] Pablo Benítez-Llambay and Frédéric S. Masset. FARGO3D: A new GPU-oriented MHD code. *The Astrophysical Journal Supplement Series*, 223(1):11, 2016.

[3] Arnaud Cassan, Daniel Kubas, J-P Beaulieu, Martin Dominik, K Horne, J Greenhill, J Wambsganss, John Menzies, Andrew Williams, Uffe Gråe Jørgensen, et al. One or more bound planets per Milky Way star from microlensing observations. *Nature*, 481(7380):167–169, 2012.

[4] Natalie M. Batalha, Jason F. Rowe, Stephen T. Bryson, Thomas Barclay, Christopher J. Burke, Douglas A. Caldwell, Jessie L. Christiansen, Fergal Mullally, Susan E. Thompson, Timothy M. Brown, Andrea K. Dupree, Daniel C. Fabrycky, Eric B. Ford, Jonathan J. Fortney, Ronald L. Gilliland, Howard Isaacson, David W. Latham, Geoffrey W. Marcy, Samuel N. Quinn, Darin Ragozzine, Avi Shporer, William J. Borucki, David R. Ciardi, Thomas N. Gautier, Michael R. Haas, Jon M. Jenkins, David G. Koch, Jack J. Lissauer, William Rapin, Gibor S. Basri, Alan P. Boss, Lars A. Buchhave, Joshua A. Carter, David Charbonneau, Joergen Christensen-Dalsgaard, Bruce D. Clarke, William D. Cochran, Brice-Olivier Demory, Jean-Michel Desert, Edna Devore, Laurance R. Doyle, Gilbert A. Esquerdo, Mark Everett, Francois Fressin, John C. Geary, Forrest R. Girouard, Alan Gould, Jennifer R. Hall, Matthew J. Holman, Andrew W. Howard, Steve B. Howell, Khadeejah A. Ibrahim, Karen Kinemuchi, Hans Kjeldsen, Todd C. Klaus, Jie Li, Philip W. Lucas, Søren Meibom, Robert L. Morris, Andrej Prša, Elisa Quintana, Dwight T. Sanderfer, Dimitar Sasselov, Shawn E. Seader, Jeffrey C. Smith, Jason H. Steffen, Martin Still, Martin C. Stumpe,

Jill C. Tarter, Peter Tenenbaum, Guillermo Torres, Joseph D. Twicken, Kamal Uddin, Jeffrey Van Cleve, Lucianne Walkowicz, and William F. Welsh. Planetary candidates observed by Kepler . III. analysis of the first 16 months of data. 204(2):24, feb 2013.

[5] M Keppler, M Benisty, A Müller, Th Henning, R van Boekel, F Cantalloube, C Ginski, R G van Holstein, A L Maire, A Pohl, M Samland, H Avenhaus, J L Baudino, A Boccaletti, J de Boer, M Bonnefoy, G Chauvin, S Desidera, M Langlois, C Lazzoni, G Marleau, C Mordasini, N Pawellek, T Stolker, A Vigan, A Zurlo, T Birnstiel, W Brandner, M Feldt, M Flock, J Girard, R Gratton, J Hagelberg, A Isella, M Janson, A Juhasz, J Kemmer, Q Kral, A M Lagrange, R Launhardt, A Matter, F Ménard, J Milli, P Mollière, J Olofsson, L Perez, P Pinilla, C Pinte, S P Quanz, T Schmidt, S Udry, Z Wahhaj, J P Williams, E Buenzli, M Cudel, C Dominik, R Galicher, M Kasper, J Lannier, D Mesa, D Mouillet, S Peretti, C Perrot, G Salter, E Sissa, F Wildi, L Abe, J Antichi, J C Augereau, A Baruffolo, P Baudoz, A Bazzon, J L Beuzit, P Blanchard, S S Brems, T Buey, V De Caprio, M Carbillet, M Carle, E Cascone, A Cheetham, R Claudi, A Costille, A Delboulbé, K Dohlen, D Fantinel, P Feautrier, T Fusco, E Giro, D Gisler, L Gluck, C Gry, N Hubin, E Hugot, M Jaquet, D Le Mignant, M Llored, F Madec, Y Magnard, P Martinez, D Maurel, M Meyer, O Moeller-Nilsson, T Moulin, L Mugnier, A Origne, A Pavlov, D Perret, C Petit, J Pragt, P Puget, P Rabou, J Ramos, F Rigal, S Rochat, R Roelfsema, G Rousset, A Roux, B Salasnich, J F Sauvage, A Sevin, C Soenke, E Stadler, M Suarez, M Turatto, and L Weber. Discovery of a planetary-mass companion within the gap of the transition disk around PDS 70. *Astronomy & Astrophysics*, 617:A44, 2018.

[6] S. Y. Haffert, A. J. Bohn, J. de Boer, I. A. G. Snellen, J. Brinchmann, J. H. Girard, C. U. Keller, and R. Bacon. Two accreting protoplanets around the young star PDS 70. *Nature Astronomy*, 3:749–754, June 2019.

[7] Ruobing Dong, Zhaohuan Zhu, Roman R Rafikov, and James M Stone. Observational signatures of planets in protoplanetary disks: Spiral arms observed in scattered light imaging can be induced by planets. *The Astrophysical Journal Letters*, 809(1):L5, 2015.

[8] Ruobing Dong, Zhaohuan Zhu, and Barbara Whitney. Observational signatures of planets in protoplanetary disks. I. gaps opened by single and multiple young planets in disks. *The Astrophysical Journal*, 809(1):93, 2015.

[9] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489, 2016.

[10] Andrew W Senior, Richard Evans, John Jumper, James Kirkpatrick, Laurent Sifre, Tim Green, Chongli Qin, Augustin Žídek, Alexander WR Nelson, Alex Bridgland, et al. Improved protein structure prediction using potentials from deep learning. *Nature*, 577(7792):706–710, 2020.

[11] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2015.

[12] Sayantan Auddy and Min-Kai Lin. A machine learning model to infer planet masses from gaps observed in protoplanetary disks. *The Astrophysical Journal*, 900(1):62, sep 2020.

[13] Sayantan Auddy, Ramit Dey, Min-Kai Lin, and Cassandra Hall. DPNNet-2.0 Part I: Finding hidden planets from simulated images of protoplanetary disk gaps, 2021.

[14] Maziar Raissi, P Perdikaris, and G E Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686 – 707, 2019-02.

[15] George Em Karniadakis, Ioannis G. Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, pages 1–19, 2021.

[16] Nicholas Geneva and Nicholas Zabaras. Modeling the Dynamics of PDE Systems with Physics-Constrained Deep Auto-Regressive Networks. *arXiv*, 403:109056, 2019.

[17] 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. *arXiv*, 394:56–81, 2019.

[18] Xiaowei Jin, Shengze Cai, Hui Li, and George Em Karniadakis. NSFnets (Navier-Stokes flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations. *Journal of Computational Physics*, 426:109951, 2021.

[19] Alireza Yazdani, Lu Lu, Maziar Raissi, and George Em Karniadakis. Systems biology informed deep learning for inferring parameters and hidden dynamics. *PLOS Computational Biology*, 16(11):e1007575, 2020.

[20] Somdatta Goswami, Cosmin Anitescu, Souvik Chakraborty, and Timon Rabczuk. Transfer learning enhanced physics informed neural network for phase-field modeling of fracture. *Theoretical and Applied Fracture Mechanics*, 106:102447, 2020.

[21] Oliver Hennigh, Susheela Narasimhan, Mohammad Amin Nabian, Akshay Subramaniam, Kaustubh Tangsali, Zhiwei Fang, Max Rietmann, Wonmin Byeon, and Sanjay Choudhry. NVIDIA SimNet™: An AI-accelerated multi-physics simulation framework. In *International Conference on Computational Science*, pages 447–461. Springer, 2021.

[22] 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.

[23] Lu Lu, Xuhui Meng, Zhiping Mao, and George E Karniadakis. DeepXDE: A deep learning library for solving differential equations. *arXiv*, cs.LG, 07 2019.

[24] Ehsan Haghighat, Maziar Raissi, Adrian Moure, Hector Gomez, and Ruben Juanes. A physics-informed deep learning framework for inversion and surrogate modeling in solid mechanics. *Computer Methods in Applied Mechanics and Engineering*, 379:113741, 2021.

[25] Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient pathologies in physics-informed neural networks, 2020.

[26] Sifan Wang, Xinling Yu, and Paris Perdikaris. When and why PINNs fail to train: A neural tangent kernel perspective, 2020.

[27] Sifan Wang and Paris Perdikaris. Long-time integration of parametric evolution equations with physics-informed DeepONets. *arXiv*, 2021.

[28] Kevin Stanley McFall and James Robert Mahan. Artificial neural network method for solution of boundary value problems with exact satisfaction of arbitrary boundary conditions. *IEEE Transactions on Neural Networks*, 20(8):1221–1233, 2009.

[29] Arthur Jacot, Franck Gabriel, and Clément Hongler. Neural Tangent Kernel: Convergence and generalization in neural networks. *arXiv*, 2018.

[30] Sifan Wang, Hanwen Wang, and Paris Perdikaris. Learning the solution operator of parametric partial differential equations with physics-informed DeepOnets. *arXiv*, 2021.

[31] Lu Lu, Pengzhan Jin, Guofei Pang, Zhongqiang Zhang, and George Em Karniadakis. Learning nonlinear operators via DeepONet based on the universal approximation theorem of operators. *Nature Machine Intelligence*, 3(3):218–229, 2021.

[32] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[33] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017.

[34] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015.

[35] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. PyTorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019.

[36] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, and Qiao Zhang. JAX: composable transformations of Python+NumPy programs, 2018.