Solving KdV Equations with Multiple Solitons using Physics Informed Neural Networks and Improved Models

Student: Mohan Ren *mohan.ren@student.manchester.ac.uk*
Department of Computer Science
The University of Manchester

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

Contents

1 Introduction **5**
1.1 Solve Partial Differential Equation using Deep Learning . . . 5
1.2 Motivation . . . 5

2 Background **8**
2.1 Partial Differential Equation . . . 8
2.1.1 Single Soliton Solution to the KdV PDE . . . 8
2.1.2 2 Soliton Solution to the KdV PDE . . . 10
2.1.3 3 Soliton Solution to the KdV PDE . . . 12
2.2 Deep Learning . . . 13
2.2.1 MLP architecture . . . 13
2.2.2 Affine Function . . . 14
2.2.3 Activation Function . . . 14
2.2.4 Gradient Descent . . . 15
2.3 Related Work . . . 16

3 Methodology **18**
3.1 PINN . . . 18
3.1.1 Problem setup . . . 18
3.1.2 Neural Network . . . 19
3.1.3 Empirical Risk Function . . . 20
3.1.4 Models with improvement . . . 20
3.1.5 Training . . . 21
3.2 Different size of DNN . . . 22

4 Results **23**
4.1 Single soliton solution of KdV experiments . . . 23
4.1.1 Models & Empirical Risk . . . 23
4.1.2 Reference Solution . . . 23
4.1.3 Results . . . 24
4.2 2-soliton solution of KdV experiments . . . 27
4.2.1 Models & Empirical Risk . . . 27
4.2.2 Reference Solution . . . 27
4.2.3 Results . . . 28
4.3 3-soliton solution of KdV experiments . . . 30
4.3.1 Models & Empirical Risk . . . 30
4.3.2 Reference Solution . . . 31
4.3.3 Results . . . 31

4.4 Experiments on "Double Descent" . . . 33

5 Reflection & Conclusion **35**
5.1 Planning . . . 35
5.2 Conclusion . . . 35

A Procedure to calculate the values of parameters of 2-soliton solution of KdV **39**

B Procedure to calculate the values of parameters of 3-soliton solution of KdV **40**

C Pseudocode for ADAM **41**

D Pseudocode for Physics Informed Neural Network **42**

List of Figures

1 One example showing the weakness of purely data-driven . . . 6
2 Both "knowledge" and "data" work together to drive the model . . . 6
3 KdV equation (Zabusky & Kruskal, 1965) . . . 9
4 Three soliton solution with $u(x,0) = 12\text{sech}^2(x)$ . . . 12
5 The basic architecture of an MLP . . . 13
6 4 Activation Functions . . . 14
7 Two cases of gradient descent: Left - the global minima is reached; Right - the local minima is reached . . . 16
8 Neural Network Example Used . . . 19
9 Double Descent (Nakkiran et al., 2019) . . . 22
10 Computational Results of 1-Soliton Solution KdV with Different Size Neural Nets . . . 25
11 Heat map of the 1-soliton solution using 3 models with 57 parameters and the exact solution: Left top: Vanilla model; Right top: Boundary-included model; Left bottom: Initial-included model; Right bottom: Exact solution . . . 26
12 Computational Results of 2-Soliton Solution KdV with Different Size Neural Nets . . . 29
13 Heat map of the 2-soliton solution using 3 models with 2109 parameters and the exact solution: Left top: Vanilla model; Right top: Boundary-included model; Left bottom: Initial-included model; Right bottom: Exact solution . . . 30
14 Results of 3-Soliton Solution KdV . . . 32
15 Left: Fractional error for the vanilla model; Right: Fractional error for the boundary-included model . . . 34

List of Tables

1 Time cost for each experiment . . . 22
2 Table of the Results of the 1-Soliton Solution from Different Models and Different P / N Ratio 24
3 Table of the Results of the 2-Soliton Solution from Different Models and Different P / N Ratio 28
4 Table of the Results of the 3-Soliton Solution from Different Models for the Comparison with the Work by Hu et al. (2022) . . . 31

Abstract

This report represents a contribution to the growing field of "AI for Science." In particular, we have reviewed recent studies involving the use of deep learning techniques to solve partial differential equations (PDEs). Our investigation has revealed that pure data-driven neural networks are not effective in solving PDEs. As a result, physics-informed neural networks (PINNs), as originally proposed by Raissi et al. (2019), have been developed and shown to deliver outstanding performance in solving PDEs in many recent works (Jagtap et al., 2020; Hu et al., 2022; Raissi et al., 2017; Cuomo et al., 2022). Moreover, according to E et al. (2021), PINNs have the potential to solve high-dimensional PDEs, which is a challenging task with traditional numerical methods like the finite element method.

However, it should be noted that vanilla PINNs do not incorporate prior knowledge such as initial and boundary conditions. The primary objective of our work was to integrate this prior knowledge with neural networks in order to improve the models' ability to fit PDEs.

In conclusion, our experiments involving the solution of single and multiple soliton Korteweg-De Vries equations using vanilla PINNs and improved models yielded several notable results. In particular, the integration of prior knowledge into the initial-included model resulted in superior performance and lower fractional error with respect to the exact solution compared to the vanilla model. The initial-included model also exhibited significant advantages in approximating the multiple solitons case and soliton intersection regions. Overall, our findings indicate that the incorporation of prior knowledge into PINN models can greatly improve their ability to fit complex PDEs.

Key Words: Machine Learning | Partial Differential Equations | Physics-Informed Neural Network | Deep Networks | Double Descent

Acknowledgement

I would like to express my sincere gratitude to my supervisor, Dr. Anirbit Mukherjee, for his invaluable support throughout this work. His knowledge, patience, encouragement, and passion for the project have been instrumental in helping me complete this report.

1 Introduction

1.1 Solve Partial Differential Equation using Deep Learning

Scientific machine learning is getting more popular and the main object is to apply AI technology to fundamental science. Partial differential equations (PDEs) are used to describe the dynamics of systems in a wide variety of science and engineering applications, therefore many recent kinds of research are focusing on how to solve PDEs using machine learning or deep learning techniques. PDEs have played important roles in many fields like physics, chemistry, engineering and finance. But many of these equations cannot be solved either analytically or even numerically. Some traditional techniques from scientific computing like the finite element method or other grid-based methods can approximate the solution well when the domain of the PDEs is low dimensional, however, these methods will fail in higher dimensions because their computational costs increase at an exponential rate, which means an explosion. Therefore, scientific machine learning is one direction now scientists going on.

Pure data-driven machine learning is not ideal to approximate the solution of PDEs because of the chaotic nature of physical phenomena described by PDEs. Therefore, a combination of the data-driven and knowledge-driven models needs to be developed so that the inferences of it can comply with physical laws. One way is to use the Physics-Informed Neural Networks (PINNs) developed by Raissi et al. (2017) and many examples from it have shown that by constraining the loss function using PDEs, the performance can be significantly increased. This is creative as PINNs can address the limitations associated with purely data-driven methodologies in the realm of scientific machine learning. Besides, it also has the potential to solve high-dimensional PDEs.

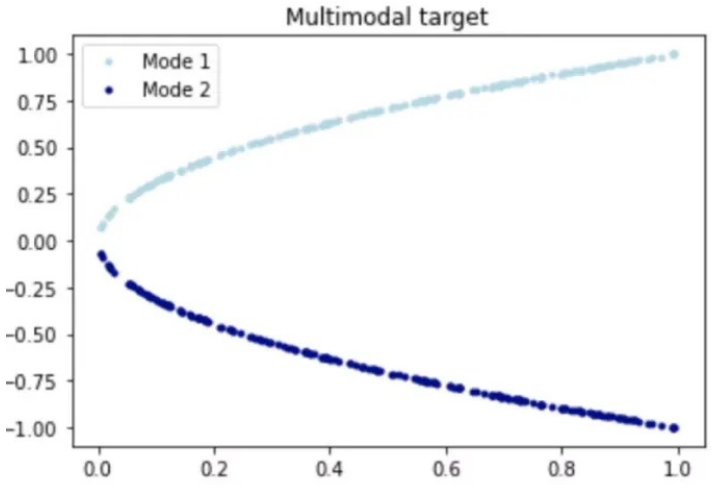
PINNs are a relatively new development in the field of PDE-solving using deep learning techniques, and over the last few years, there have been many advancements in this direction. However, vanilla PINNs do not have the ability to integrate prior or known conditions, such as initial or boundary conditions, for a general form of PDE. In these cases, PINNs simply add these conditions as penalty terms in the empirical loss function. As a result, our work has focused on integrating prior knowledge into the model, resulting in the development of two new models: the initial-included model and the boundary-included model.

In this work, we selected the Korteweg-De Vries (KdV) equation as our target PDE due to its unique characteristic of having multiple soliton solutions despite possessing the same PDE equation. Additionally, the KdV equation is a classic example of where vanilla PINNs can fail. Our experiments involved solving single soliton, double soliton, and triple soliton cases of the KdV equation, and through comparisons of the two models we proposed, we determined that integrating the initial condition into vanilla PINNs resulted in significantly better performance for all soliton cases. Furthermore, we observed a trade-off between time cost and accuracy in the single soliton and double soliton cases, indicating that time efficiency should be considered in selecting the best model.

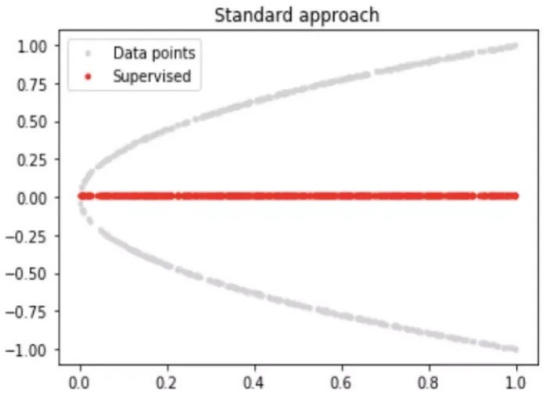
Given that "Double Descent" is a recently discovered phenomenon used to describe how large-scale neural networks can impact machine learning tasks, we investigated whether it also applies to PDE-solving problems using PINNs. Following our experiments, we did not find strong evidence suggesting that "Double Descent" occurs in PDE-solving problems involving PINNs.

1.2 Motivation

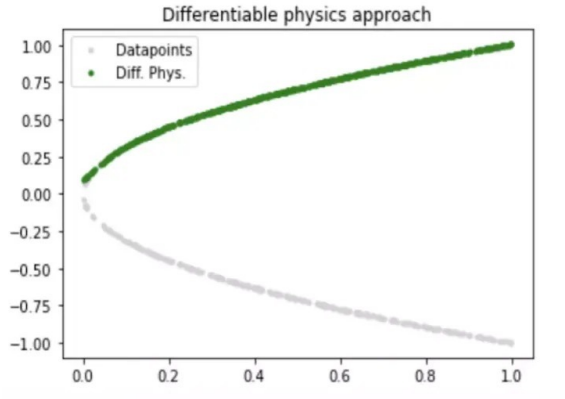
PDE is one of the most important tools to describe the physical world, and machine learning especially deep learning techniques is gaining huge popularity. In the past years, many researchers hope to use deep learning to describe our physical world and some did great success in visual recognition or natural language processing fields. However, as a pure data-driven model, deep learning cannot be efficiently applied to scientific topics. Purely data-driven machine learning models might be well-suited for applications dealing with large amounts of observational data; however, the generalization performance may suffer due to extrapolation or observation bias, leading to inferences that do not comply with physical laws and are thus deemed unreliable. Another drawback stems from the chaotic nature of physical phenomena, such as bifurcation events (Bifurcation). Let's examine a relatively simple example to illustrate the weakness of purely data-driven approaches in this aspect: Fitting $y^2 = x$ (shown in 1(a)), which is actually a representation of a parabolic function ($y = x^2$) lying horizontally.



(a) Results of $y^2 = x$



(b) Results for fitting use only data-driven model



(c) Results for adding physics-driven

Figure 1: One example showing the weakness of purely data-driven

If we sample from the parabolic curve (x_i, y_i) and deploy a neural net to fit: $\min_\theta \sum_i \mathcal{L}(y_i, \hat{y}_\theta(x_i))$, we probably have the line in figure 1(b). Despite the fact that this red line may possess some statistical relevance for a particular phenomenon, it fails to satisfy the essential physical constraints and, as a consequence, its reliability is questionable.

Another way is adding some physics information to make the model physics-driven. For example, the figure 1(c) shows the results for fitting by minimizing: $\min \sum_i |y_\theta(x_i)^2 - x_i^2|^2$. And this is the so-called "Physics Informed".

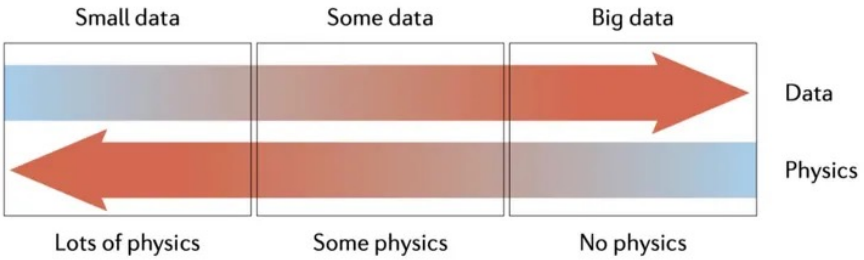


Figure 2: Both "knowledge" and "data" work together to drive the model

This is the motivation to research the fusion approach combining data-driven and knowledge-driven methods, which is the central region of the figure 2.

Six years ago, Raissi et al. (2017) first introduced the concept of "Physics Informed Neural Networks" (PINN) on Arxiv. Since then, numerous studies from the academic community have been conducted based on this pioneering work. Many experiments have shown the power of PINNs and great performance on many famous PDEs like Navier–Stokes equation[(Raissi et al., 2019)], Poisson equation[(Hu et al., 2022)]. However, there are some doubts from Grossmann et al. (2023) that whether PINNs can beat the finite element method and others (Krishnapriyan et al., 2021; Wang et al., 2021) gave their analysis on what case the PINNs may fail.

The Korteweg-De Vries (KdV) equation is a highly significant equation in the study of wave motion as it employs the concept of "soliton" to describe wave behaviour. Numerous researchers have experimented with solving KdV equations involving single or double solitons by PINNs, but only a few have attempted to find solutions with more solitons. Moreover, as a nonlinear PDE, the KdV equation can potentially become a failure mode for PINN (Krishnapriyan et al., 2021), as observed later in figure 14(b) when attempting to solve a 3-soliton KdV.

Upon analyzing the principles of the vanilla PINNs, it becomes apparent that the model does not incorporate information from boundaries or initial conditions, resulting in unused known knowledge. Consequently, there exists a need to develop new models capable of integrating these known conditions, with the expectation that at least one will outperform vanilla PINNs for solving KdV equation.

All of the aforementioned factors inspired me to continue the research on PINN by solving KdV equations with multiple solitons. Additionally, this work spurred my interest in comprehending the potential of scientific machine learning. Given the extensive use of PDEs in numerous fields such as physics, finance, and engineering, discovering solutions through artificial intelligence techniques would significantly facilitate interdisciplinary connections and contribute to the advancement of computer intelligence in a cyclical manner.

My passion for artificial intelligence leads me to seek out complex problems, particularly in the realm of partial differential equations. I find great satisfaction in leveraging my mathematical knowledge and AI expertise to explore and unravel these intricate issues. Developments in AI have provided new and innovative solutions that allow us to tackle some of the most difficult challenges in this field with ease. Using a combination of experience, knowledge, and skill, I am committed to pushing the boundaries of what is possible and using these emerging technologies to generate impactful change in real-world applications.

2 Background

This section aims to provide fundamental knowledge about partial differential equations (PDEs) and deep learning. It begins with an exposition of PDEs, delving into their historical significance, followed by an overview of the Korteweg-de Vries equation - a renowned PDE referenced extensively throughout the experiments. Additionally, the architecture of the multilayer perceptron alongside the activation function and affine transformation will be introduced, aiding in a review of basic concepts central to deep learning.

After these, the literature review will include influential studies in the field of PDE-solving, highlighting key developments while also pointing to areas that warrant further investigation. By presenting this expanded context for the topic, readers are spurred to broaden their understanding of related research and invigorate their own experimentation in this active and ever-evolving domain.

2.1 Partial Differential Equation

Partial Differential Equations (PDEs) have a rich history in modelling physical and biological phenomena, dating back to the laws of fluid motion and heat conduction formulated by Isaac Newton. In the 19th century, mathematicians such as Cauchy, Laplace, and Riemann made significant contributions to the theory of PDEs, paving the way for modern mathematical treatments of these equations. The 20th century saw the introduction of Hilbert spaces and weak solutions, which further expanded the scope and depth of PDE theory, and contributed to their widespread use in diverse fields of study.

Nowadays, PDEs find applications in various fields such as engineering, finance, and imaging science. They serve as mathematical models for complex systems in diverse areas like fluid dynamics, quantum mechanics, and general relativity. Furthermore, PDEs play a pivotal role in many modern technologies, including weather forecasting, medical imaging, and computer graphics. As such, the study of PDEs continues to be an essential component of many scientific and technological endeavours. Below gives the definition of PDE:

Definition 1 (PDE, solution)**.** *A partial differential equation is an equation containing an unknown function (say, u) of two or more variables and its partial derivatives with respect to these variables. A solution is a function which, when substituted for u, satisfies the PDE.*

The example equation

$$\frac{\partial u}{\partial t} + \frac{\partial^2 u}{\partial x^2} = 0$$

is a PDE. And the function $u(x,t) = 6t - 3x^2$ is a solution.

Usually, a PDE will not have a unique solution. In fact, there will be infinitely many solutions. Furthermore, no all PDEs has exact solutions and one can be the PDE without solution. For example,

$$\begin{cases} x\dfrac{\partial u}{\partial t} + \dfrac{\partial u}{\partial x} = 0 \\ u(x,0) = \sin(x). \end{cases}$$

Conditions such as $u(x,0)$ where t represents time are called *initial conditions*. Conditions such as $u(0,t) = f(t)$ where x represents space are called *boundary conditions*.

Our experiments centered on the Korteweg-de Vries equation (KdV), a unique PDE that presents with diverse solutions contingent on the number of solitons. As an important, legendary, and historically significant equation, KdV has been the subject of extensive study due to its intriguing phenomena. Thus, this section aims to furnish readers with a broad-based background and significance of the KdV equation, reinforcing their understanding of its implications within the field of partial differential equations.

2.1.1 Single Soliton Solution to the KdV PDE

The Korteweg–de Vries Equation (KdV) is among the most significant PDEs in the world as it describes the behaviour of small-amplitude, long-wavelength waves in various physical systems such as shallow water waves. As noted by de Jager (2006), KdV's history dates back about sixty years, beginning with Scott

Russell's experiments in 1834, followed by theoretical investigations of notable figures such as Lord Rayleigh and Boussinesq in 1871, and ultimately Korteweg and De Vries in 1895.

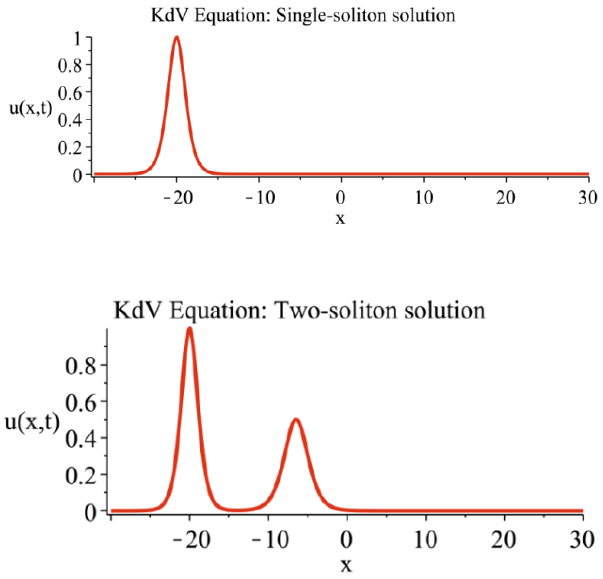


Figure 3: KdV equation (Zabusky & Kruskal, 1965)

After its initial discovery, the KdV equation remained relatively unchanged until Zabusky & Kruskal (1965) made a significant numerical discovery that its solutions appeared to decompose into a collection of "solitons" at large times. Solitons are unique types of waves that maintain their shape and velocity as they propagate through a medium. Figure 3 displays images of waves at a fixed time point, with the upper figure showing a plot of a single soliton KdV and the lower figure depicting a plot of two solitons KdV.

The KdV equation is a nonlinear, dispersive partial differential equation for a function u of two dimensionless real variables, x and t which are proportional to space and time respectively:

$$\begin{cases} \dfrac{\partial u}{\partial t} + \dfrac{\partial^3 u}{\partial x^3} + 6\,u\,\dfrac{\partial u}{\partial x} = 0, (x,t) \in \Omega \times T, \\ u(x,0) = u_0(x), x \in \Omega, \\ u(x,t) = g(x,t), (x,t) \in \partial\Omega \times T, \\ \Omega, T \subset \mathbb{R} \text{ both bounded} \end{cases} \tag{1}$$

where $u_0(x)$ is the initial condition and $g(x,t)$ is the boundary condition.

Regarding the equation itself, the constant 6 in front of the last term is simply a convention and does not hold great significance. Multiplying t, x, and u by constants can be used to make the coefficients of any of the three terms equal to any given non-zero constants.

Below are the solution (Taha & Ablowitz, 1984) and the proof of the single soliton solution of the KdV equation:

$$u(x,t) = 2\,\mathrm{sech}^2(x - 4t - \eta_0) \tag{2}$$

Proof. Define $X \coloneqq x - 4t - \eta_0$

$$
\begin{aligned}
\frac{\partial u}{\partial t} &= -16\tanh(-X)\operatorname{sech}^2(-X) \\
\frac{\partial u}{\partial x} &= 4\tanh(-X)\operatorname{sech}^2(-X) \\
\frac{\partial^3 u}{\partial x^3} &= 16\tanh(-X)\operatorname{sech}^2(-X)(-2\operatorname{sech}^2(-X)+\tanh^2(-X))
\end{aligned}
$$

$$
\begin{aligned}
&\partial_t u + \partial_x^3 u + 6\,u\,\partial_x u \\
=& -16\tanh(-X)\operatorname{sech}^2(-X) \\
&+16\tanh(-X)\operatorname{sech}^2(-X)(-2\operatorname{sech}^2(-X)+\tanh^2(-X)) \\
&+6(2\operatorname{sech}^2(X))(4\tanh(-X)\operatorname{sech}^2(-X)) \\
=&16\tanh(-x)\operatorname{sech}^2(-x) \\
&\times(-1+3\operatorname{sech}^2(-X)-\tanh^2 X-2\operatorname{sech}^2 X) \\
=&0
\end{aligned}
$$

□

2.1.2 2 Soliton Solution to the KdV PDE

Given the same KdV PDE 1, the solutions are different according to the different numbers of solitons. For the two-soliton KdV, the solution (Taha & Ablowitz, 1984) on the infinite interval is:

$$u(x,t) = 2(\log f)_{xx} \tag{3}$$

where

$$
\begin{aligned}
f &= 1 + e^{\eta_1} + e^{\eta_2} + e^{\eta_1+\eta_2+A_{12}} \\
\eta_i &= k_i x - k_i^3 t + \eta_i^{(0)}
\end{aligned}
$$

and

$$e^{A_{ij}} = (\frac{k_i - k_j}{k_i + k_j})^2.$$

Below gives the proof that this is indeed the solution of KdV:

Proof.

$$2(\log f)_{xx} = \frac{f \times f'' - (f')^2}{f^2}$$

$$
\begin{aligned}
f &= 1 + e^{\eta_1} + e^{\eta_2} + e^{\eta_1+\eta_2+A_{12}} \\
f' &= k_1 \times e^{\eta_1} + k_2 \times e^{\eta_2} + (k_1+k_2)e^{\eta_1+\eta_2+A_{12}} \\
f'' &= k_1^2 \times e^{\eta_1} + k_2^2 \times e^{\eta_2} + (k_1+k_2)^2 \times e^{\eta_1+\eta_2+A_{12}}
\end{aligned}
$$

$$
\begin{aligned}
&u(x,t)\\
=&2\times\frac{f\times f''-(f')^2}{f^2}\\
=&2\times\frac{(1+e^{\eta_1}+e^{\eta_2}+e^{\eta_1+\eta_2+A_{12}})\times(k_1^2\times e^{\eta_1}+k_2^2\times e^{\eta_2}+(k_1+k_2)^2\times e^{\eta_1+\eta_2+A_{12}})-(k_1\times e^{\eta_1}+k_2\times e^{\eta_2}+(k_1+k_2)\times e^{\eta_1+\eta_2+A_{12}})^2}{(1+e^{\eta_1}+e^{\eta_2}+e^{\eta_1+\eta_2+A_{12}})^2}\\
=&2\times\frac{k_1^2e^{\eta_1}+k_2^2e^{\eta_2}+(k_1+k_2)^2e^{\eta_1+\eta_2}+(k_1+k_2)^2e^{\eta_1+\eta_2+A_{12}}+k_2^2e^{2\eta_1+\eta_2+A_{12}}+k_1^2e^{\eta_1+2\eta_2+A_{12}}}{1+2e^{\eta_1}+2e^{\eta_2}+2e^{\eta_1+\eta_2+A_{12}}+e^{2\eta_1}+e^{2\eta_2}+2e^{\eta_1+\eta_2}+2e^{2\eta_1+\eta_2+A_{12}}+2e^{\eta_1+2\eta_2+A_{12}}+e^{2\eta_1+2\eta_2+2A_{12}}}
\end{aligned}
$$

To simplify, let

$$A=k_1^2e^{\eta_1}+k_2^2e^{\eta_2}+(k_1+k_2)^2e^{\eta_1+\eta_2}+(k_1+k_2)^2e^{\eta_1+\eta_2+A_{12}}+k_2^2e^{2\eta_1+\eta_2+A_{12}}+k_1^2e^{\eta_1+2\eta_2+A_{12}}$$

$$B=1+2e^{\eta_1}+2e^{\eta_2}+2e^{\eta_1+\eta_2+A_{12}}+e^{2\eta_1}+e^{2\eta_2}+2e^{\eta_1+\eta_2}+2e^{2\eta_1+\eta_2+A_{12}}+2e^{\eta_1+2\eta_2+A_{12}}+e^{2\eta_1+2\eta_2+2A_{12}}$$

Now have:

$$
\begin{aligned}
u(x,t)&=2\frac{A}{B}\\
\frac{\partial u}{\partial t}&=2\frac{\partial_t AB-A\partial_t B}{B^2}\\
\frac{\partial u}{\partial x}&=2\frac{\partial_x AB-A\partial_x B}{B^2}\\
\frac{\partial^2 u}{\partial x^2}&=2\frac{(\partial_{xx}AB-A\partial_{xx}B)B+(\partial_x AB-A\partial_x B)2\partial_x B}{B^3}\\
\frac{\partial^3 u}{\partial x^3}&=2\frac{(\partial_{xxx}AB+\partial_{xx}A\partial_x B-\partial_x A\partial_{xx}B-A\partial_{xxx}B)B^2}{B^4}\\
&+2\frac{(\partial_x AB-A\partial_x B)(2B\partial_{xx}B-6(\partial_x B)^2)}{B^4}
\end{aligned}
$$

$$
\begin{aligned}
&\partial_t u+\partial_x^3 u+6\,u\,\partial_x u\\
=&2\frac{\partial_t AB-A\partial_t B}{B^2}\\
&+2\frac{(\partial_{xxx}AB+\partial_{xx}A\partial_x B-\partial_x A\partial_{xx}B-A\partial_{xxx}B)B^2}{B^4}\\
&+2\frac{(\partial_x AB-A\partial_x B)(2B\partial_{xx}B-6(\partial_x B)^2)}{B^4}\\
&+24\frac{A}{B}\times\frac{\partial_x AB-A\partial_x B}{B^2}\\
=&0
\end{aligned}
$$

□

According to Drazin & Johnson (1989), in the two solitons case, there is a moment where there is just one pulse at $t=0$. By using the initial condition given by their book:

$$u(x,0)=N(N+1)\mathrm{sech}^2(x), \tag{4}$$

we can calculate the values of the parameters appearing in the solution: $k_1, k_2, \eta_1^0, \eta_2^0$ which decide the moves of the waves we expect.

The steps to find the values:$k_1 = -2, k_2 = 4, \eta_1^0 = \ln\frac{1}{3}, \eta_2^0 = \ln\frac{1}{3}$ are listed in Appendix A. For the forthcoming experiments on the 2-soliton case, it is intended to employ the four parameters available to achieve a greater level of solution consistency.

2.1.3 3 Soliton Solution to the KdV PDE

Same as we did in 2.1.2, below are the three solitons solution of KdV 1 given before:

$$u(x,t) = 2(\log f)_{xx} \tag{5}$$

where

$$f = 1 + e^{\eta_1} + e^{\eta_2} + e^{\eta_3} + e^{\eta_1+\eta_2+A_{12}} + e^{\eta_2+\eta_3+A_{23}} + e^{\eta_3+\eta_1+A_{31}} + e^{\eta_1+\eta_2+\eta_3+A_{12}+A_{23}+A_{31}}$$

$$\eta_i = k_i x - k_i^3 t + \eta_i^{(0)}$$

and

$$e^{A_{ij}} = (\frac{k_i - k_j}{k_i + k_j})^2.$$

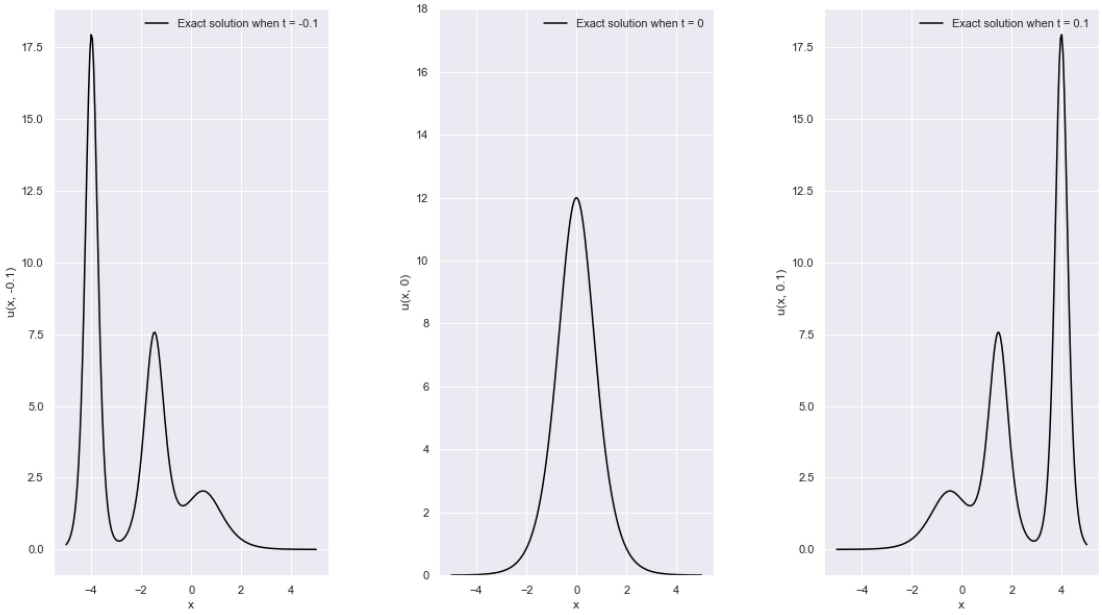


Figure 4: Three soliton solution with $u(x,0) = 12\text{sech}^2(x)$

The proof of the solution is similar to the two solitons case and therefore we will not repeat it here. Again, according to Drazin & Johnson (1989), in the three solitons case, there is a moment where there is just one peak at $t = 0$ shown in 4. The initial condition to satisfy the condition is given by equation 4. By consistantly using this initial condition, we can see that one phenomenon of KdV is that when the initial wave is higher, it can seperate into many small waves, and higher the initial wave is, more sub-waves it will have when time flows.

To make the solution more consistent (only one peak at $t = 0$ to match the initial condition equation 4, which are we always did in all KdV experiments), we need to calculate the value of 6 parameters appearing in the solution: $k_1, k_2, k_3, \eta_1^0, \eta_2^0, \eta_3^0$ and then we can see the same situation in our code output.

Overall we calculated a set of values of them: $k_1 = 2, k_2 = 4, k_3 = -6, \eta_1^0 = \ln\frac{3}{2}, \eta_2^0 = \ln\frac{3}{5}, \eta_3^0 = \ln\frac{1}{10}$ and procedure for the calculation is available in the Appendix B.

Now we can see that by varying the initial condition, different numbers of solitons can be implemented for the KdV equation and that is one magic point of it.

2.2 Deep Learning

Traditionally, physical phenomena described by partial differential equations (PDEs) have been resolved through traditional physics computational techniques. However, as highlighted by Ray et al. (2023), fundamental disparities exist between such classic physics computation methods and machine learning techniques. To facilitate comprehension of our experiments, this section aims to elucidate several key concepts intrinsic to the realm of machine learning.

Besides, at times the terms machine learning and deep learning are used interchangeably, but in reality, they are related but different stuff and deep learning is the subset of machine learning. The simplest form of DL architecture is the feed-forward network, which comprises several layers of non-linear transformation. The non-linear transformations are applied to an affine transformation (2.2.2) of an intermediate output.

Therefore, now we shall learn or recap some basic backgrounds of deep learning.

2.2.1 MLP architecture

The concept of "Perceptron" was first developed by Rosenblatt (1958) in 1958 and his work shows how information from the physical world can be sensed. The MLP means multilayer perceptron and is the main technique needed in this report or all deep learning tasks.

A Multi-Layer Perceptron (MLP) typically consists of at least three layers of nodes, including the input layer, hidden layers, and output layer. Each node in the MLP, except for those in the input layer, can be viewed as a neuron that performs nonlinear transformations using an activation function. The MLP utilizes a back-propagation algorithm, which employs chain rules, to train its internal parameters. The basic architecture of an MLP is illustrated in Figure 5.

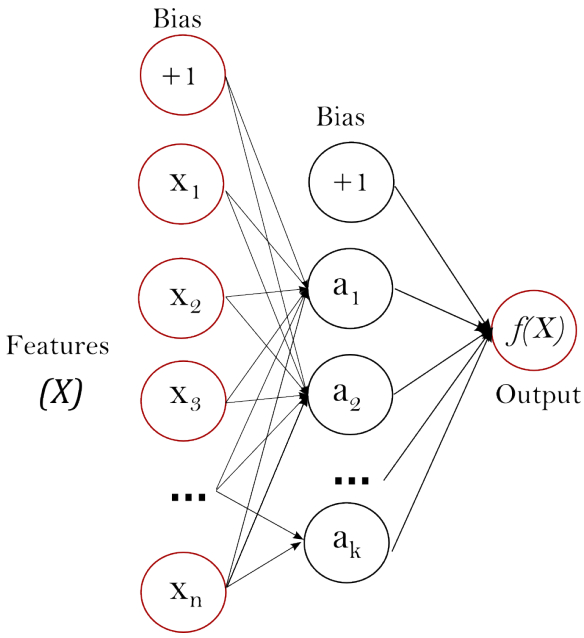


Figure 5: The basic architecture of an MLP

The figure above clearly shows how an MLP works. When features, say X, are offered in a vector form, they will be operated by the hidden layer using weights and bias which are trainable parameters, and then output the $f(X)$. Among the operations, the affine transformation 2.2.2 and activation 2.2.3 are the most used techniques. If an affine function is defined as $\mathrm{A} : \mathbb{R}^m \to \mathbb{R}^n$ and define activation function as σ, then the $f(X)$ above is equal to $f(X) = \sigma \circ \mathrm{A}(X)$.

2.2.2 Affine Function

The affine transformation is often mistakenly interpreted as a linear transformation, which is incorrect and leads to misunderstandings. The form of the affine transformation allows for an offset on top of a linear transformation. Below is the functional expression of the affine transformation:

$$\mathrm{A}(\vec{x}) = \boldsymbol{W}(\vec{x}) + \vec{b},$$

where $\boldsymbol{W}$ denotes the weight matrix in neural nets and $\vec{b}$ denotes the bias vector.

In each fully connected layer of the neural network, an affine transformation is applied to perform computations. By utilizing gradient descent-related algorithms such as Adam (Kingma & Ba, 2017), the weight and bias parameters can be optimized to achieve an ideal transformation. The goal of this optimization process is to enable the output after the affine transformation, using the optimized weight and bias, to approximate the true values as closely as possible.

2.2.3 Activation Function

Activation functions play a crucial role in the success of neural networks, introducing nonlinearity and enabling more complex modelling of input-output relationships. There are many different activation functions available, including well-known choices such as sigmoid, ReLU, and Tanh, each with their strengths and weaknesses. In a recent literature review by Bhoi et al. (2021), the most commonly used activation functions in deep neural networks are analyzed in detail, providing insights into the benefits and drawbacks of each approach. This research highlights the importance of choosing the appropriate activation function for specific network architectures and tasks.

Some activation gates applied in this report will be seen in the following contents. Besides, all activations used in DNN are defined as σ

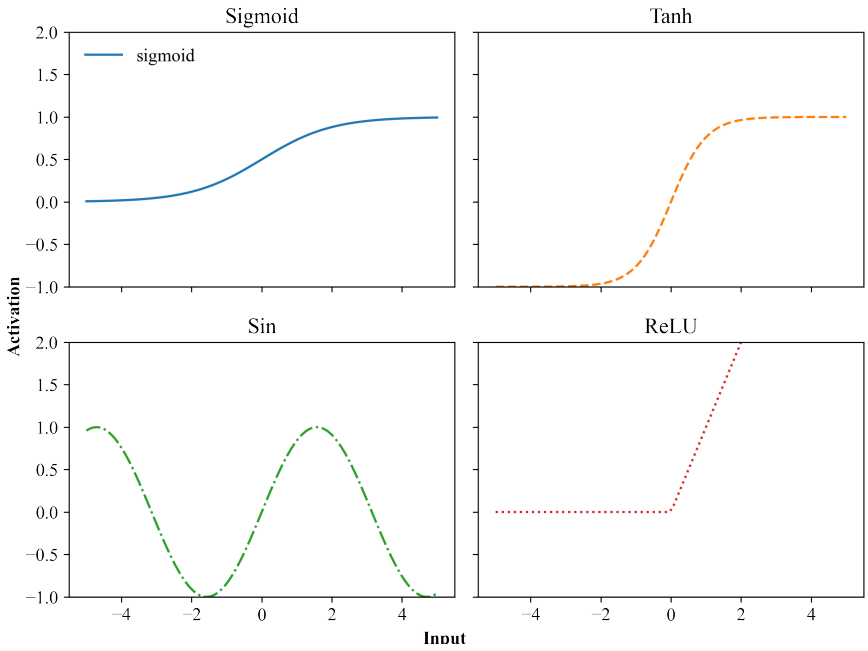


Figure 6: 4 Activation Functions

Sigmoid Function A classic activation with an "S"-shape curve(figure 6). The function is infinitely smooth and monotonic. It is defined by:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Tanh Function A function with a similar shape as "S" but the range of output is larger than the sigmoid function. The function is infinitely smooth (figure 6). It is defined by:

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

Sin Function Another common activation used for solving PDE. The function is infinitely smooth (figure 6). It is defined by:

$$\sigma(x) = \sin(x)$$

ReLU Function The most famous and popular activation in deep learning tasks. The function is continuous and piecewise (figure 6). It is defined by:

$$\sigma(x) = \max(0, x)$$

It should be noted that when differentiating an equation, the target function must be smooth within the domain. However, it is important to recognize that ReLU activation functions do not necessarily satisfy this requirement as they are not differentiable at zero.

Selecting an appropriate activation function is a non-trivial task, as each function has unique characteristics and behaviours. In the forthcoming experiments, the Sigmoid activation function will be utilized to solve most cases.

2.2.4 Gradient Descent

Recall the purpose of the neural network. Assume that we are given a dataset of pairwise samples $\mathcal{D} = \{(x_i, y_i) : 1 \le i \le N\}$ corresponding to a target neural net function $\mathcal{N} : x \to y$, we wish to approximate the function using the net $\mathcal{N}(x; \theta, \Theta)$ where θ are the network parameters defined, while Θ corresponds to the hyper-parameters of the network such as the depth or the width or the activation function, etc. And then the aim of training a neural network $\mathcal{N}$ is finding:

$$\theta^* = \arg\min_{\theta} \prod(\theta),$$

where $\prod(\theta) = \frac{1}{N}\sum_i^N \|y_i - \mathcal{N}(x_i; \theta, \Theta)\|^2$ for some fixed Θ. In this process, gradiend descent is then the algorithm used to find the optimal θ^*.

An overview paper about gradient descent made by Ruder (2017) has demonstrated this optimization method and the core idea of the vanilla batch gradient descent is:

$$\theta = \theta - \eta \nabla_\theta \hat{\mathcal{R}}(\theta),$$

where η is the learning rate or step length and $\hat{\mathcal{R}}$ is the empirical risk. To increase the speed of the descent, there are two ways "Stochastic GD" and "Mini-batch GD" that are faster but have a reasonable high variance.

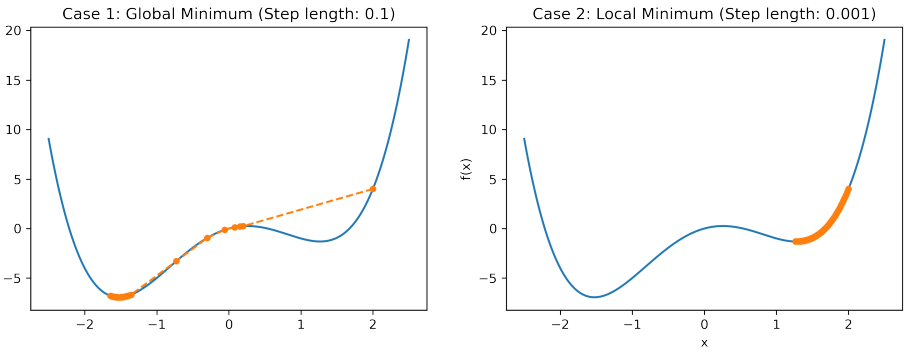


Figure 7: Two cases of gradient descent: Left - the global minima is reached; Right - the local minima is reached

The ideal outcome for optimization is to achieve optimized θ^* at the global minima; nevertheless, neural network empirical risk functions' non-convexity necessitates the consideration that numerous suboptimal local minima may also be present, leading descent astray. While such suboptimal solutions may still be viable in certain situations, identifying the global minima is essential when formulating effective models capable of taking on complicated observable phenomena gleaned from experimental data.

Figure 7 depicts the final results of Gradient Descent (GD) for two cases, with step length η playing a crucial role in determining outcomes. Recognizing the challenge gradient descent presents has led to the development of optimized algorithms, such as Adagrad (Duchi et al., 2011), RMSprop, and Adam (Kingma & Ba, 2017). Many of these algorithms operate using adaptive step lengths, demonstrating improved performance over traditional GD methods. A pseudocode representation of the Adam algorithm can be found in Appendix C.

The Adam optimizer is widely employed in deep learning tasks, hence its prevalence in experiments throughout this report. However, it is worth noting that for training PINNs, some studies have indicated suboptimal performance with the Adam algorithm. In response to this, alternative optimization strategies have been developed, such as the AGDA (Adaptive Gradient Descent Acceleration) technique, proposed by Liu et al. (2022). Careful consideration of these findings can yield better results when performing tasks involving Physics-informed Neural Networks.

2.3 Related Work

In this section, we will mainly discuss the related works for solving PDE using Deep learning techniques.

At this time, "AI for science" is getting more and more popular and a huge number of the latest deep learning techniques have been applied to the research in basic subjects like physics, chemistry, etc. From E et al. (2021)'s review paper on algorithms for solving high-dimensional PDEs, we have seen many example methods like Monte Carlo and Deep Ritz method (E & Yu, 2017). Deep learning has emerged as a powerful tool for solving PDEs. They can learn the underlying physics of PDEs directly from data, without the need for explicit knowledge of the underlying equations Ray et al. (2023).

One of the earliest works in this area is the Deep Galerkin Method (DGM) from Sirignano & Spiliopoulos (2018). The DGM uses a neural network to represent the solution of a PDE and trains it to minimize the residual error of the PDE. The method was tested on several benchmark problems, including the heat equation and the Navier-Stokes equation, and showed better accuracy and efficiency than traditional numerical methods.

Later, a similar approach called Physics Informed Neural Network (PINN) (Raissi et al., 2019) is proposed in 2019, which combined DL with physical principles. In his work, PINN has been applied to several PDE problems, including the Burgers' equation and the Schrödinger equation, and has shown to be accurate and efficient. Nowadays PINN is the most important and popular approach for solving PDEs using neural networks and some variations like XPINN (Hu et al., 2022) or CPINN (Jagtap et al., 2020) were also

developed to improve the performance of the vanilla PINN. Moreover, many analysis work on PINN were also achieved by Grossmann et al. (2023); Mishra & Molinaro (2022); Wang et al. (2021). However, the PINN is not always perfect when facing some complicated cases like convection, reaction and diffusion operators. Therefore, Krishnapriyan et al. (2021)'s work analyzed possible failure modes in PINNs. Grossmann et al. (2023) also critically analyzed the PINN by comparing the performance with the traditional finite element method.

In addition to these, there are other notable works in this area, such as the Equation Learning Network (EQN), proposed by Wang et al. (2022). EQN uses a neural network to learn the operator of a PDE directly from data, without the need for explicit knowledge of the underlying PDE. The method was shown to be effective in discovering the hidden structure of PDEs.

Another interesting work is the Neural Operator method, proposed by Kovachki et al. (2022). The Neural Operator combines a neural network with traditional numerical methods to solve PDEs. The method involves training a neural network to learn the nonlinear part of a linear operator, while the remaining part is solved using traditional methods. The method was shown to be accurate and efficient for various PDE problems.

Even though there are many kinds of methods, most jobs are still processed around the PINNs. The review from Cuomo et al. (2022) shows where we are and what's next for the research on PINNs, and clearly, we noticed that most jobs are using different activation functions, optimization algorithms, neural net structures and losses to customize the PINN. The current demand for PINNs is still a theoretical question as it is unsolved by comparing with numerical methods like the finite element method.

As PINNs are rapidly being developed, several software packages, including DeepXDE (Lu et al., 2021), NVIDIAModulus (previously SimNet) (Hennigh et al., 2021), PyDEns (Koryagin et al,2019) (Koryagin et al., 2019), and NeuroDiffEq (Chen et al., 2020) were released in 2019 to make training PINNs easier and faster.

Overall, the use of PINNs in solving PDEs is a rapidly evolving area of research with significant potential for practical applications. The ability to combine physical principles with deep learning offers new opportunities for solving complex PDE problems efficiently and accurately. With further research and development, PINNs will likely become even more powerful tools for solving PDEs in various fields such as physics, engineering, and finance.

3 Methodology

3.1 PINN

According to Raissi et al. (2019), Physics Informed Neural Network (PINN) is a kind of deep learning technique that can be trained to solve supervised learning tasks while respecting any given laws of physics described by general nonlinear partial differential equations. For a simple introduction to PINNs, the principle behind Physics-Informed Neural Networks (PINN) involves employing neural networks to approximate solutions of partial differential equations (PDEs) by minimizing a loss function. The loss function comprises residuals for initial and boundary conditions as well as partial differential equation residuals at selected points within the domain.

The idea of PINN is not very novel. Generally, in textbooks on numerical analysis, more emphasis is given to grid-based methods such as finite difference, finite element, and finite volume methods. Among them, the finite difference method is the most basic method, which uses a difference quotient to approximate derivatives in space, and discrete-time steps to advance in time. The finite element method can adaptively partition irregularly shaped regions, and use some basic functions to approximate analytical solutions, while the finite volume method can handle "conservative" problems, such as material transport and electromagnetic field, where physical quantities are usually conservative, that is, there exists a conservative law in the system.

Although grid-based methods are widely used in scientific and engineering computing, they also have some drawbacks. For example, they may be affected by boundary conditions and initial conditions and may require more computational resources when grid partitioning becomes difficult in high-dimensional space. In addition, grid-based methods usually cannot handle some special cases well, such as unclear equations of state or dissipative problems.

There is also another class of methods opposite to grid-based methods, namely so-called meshless methods, where we can find the prototype of PINN (Lawal et al., 2022). One of the simplest meshless methods based on strong-form radial basis functions (RBF) is Kansa's method (Pang et al., 2015), while RBF is already known as a shallow neural net developed by Broomhead & Lowe (1988).

However, when it comes to learning complex features, shallow neural networks may be insufficient, and increasing network width can make the linear system extremely ill-conditioned. Moreover, selecting hyperparameters for basis functions and choosing collocation points is also challenging for meshless methods Sukumar & Srivastava (2022). Therefore, given the limitations of a one-layer RBF network, an intuitive and logical next step would be to transition to a more complex architecture such as a multi-layer perceptron (MLP). It is this extension that served as the catalyst for the formulation of the PINN paradigm.

If we consider the Physics-Informed Neural Network (PINN) as a mere numerical solver, it generally cannot compete with conventional methods (including finite difference, finite element, and finite volume methodologies) in terms of speed or accuracy (Grossmann et al., 2023). Nonetheless, the growing popularity of this approach must be attributed to its unique characteristics. One advantage of PINNs is that they can address the limitations associated with purely data-driven methodologies in the realm of scientific machine learning. If traditional numerical schemes are viewed as solely driven by physical knowledge, then PINNs, or more broadly, machine learning with embedded physics knowledge, represent a synergistic combination of data-driven and knowledge-driven paradigms (Ray et al., 2023).

Additionally, in the past several years, a series of numerical experiments made by Mishra & Molinaro (2022) also estimated the upper bound of the generalization error for PINN, giving some rationale for why PINNs are so efficient at approximating solutions for PDEs.

3.1.1 Problem setup

To show the general form of PDEs that can be solved by PINN, below we give the problem setup.

Consider a parametrized and nonlinear PDE with d spatial dimensions:

$$\frac{\partial u(x,t)}{\partial t} + \mathcal{L}[u(x,t);\lambda] = 0, (x,t) \in \Omega \times [0,T] \tag{6}$$

where $u(x,t)$ denotes the latent (hidden) solution, $\mathcal{L}[\cdot\,;\lambda]$ is a nonlinear operator parametrized by λ, and Ω is a subset of $\mathbb{R}^d$.

The general form described is applicable to various types of partial differential equations (PDEs), encompassing an array of mathematical physics laws such as conservation laws, diffusion processes, and advection-diffusion-reaction systems. In this report, the Korteweg-de Vries equation shall be utilized as a prominent exemplar pertinent to the investigation of soliton dynamics. Besides, we can see that KdV equation 1 can be transformed to the general form equation 6 used by PINNs.

Furthermore, Appendix D provides a lexicon of pseudocode outlining the implementation process for the PINNs employed throughout this study.

3.1.2 Neural Network

In the context of this report, a fully connected neural network has been developed to resolve the PDE in question. Following an overview of the affine transformation 2.2.2 and activation function 2.2.3 discussed in previous sections, the architecture of our utilized neural network can be subsequently presented. The activation function is defined as σ while the affine transformation is defined as A. Furthermore, for this PDE-solving problem, there are only two inputs x and t and one output $u(x,t)$, therefore the dimension of the neural net's input and output gates are able to be fixed.

To make the network definition more general, we define the affine transformation used:

$$\mathrm{A}_1 : \mathbb{R}^2 \to \mathbb{R}^n, \mathrm{A}_2, ..., \mathrm{A}_{m-1} : \mathbb{R}^n \to \mathbb{R}^n, \mathrm{A}_m : \mathbb{R}^n \to \mathbb{R},$$

where m is the depth of the net and n is the width of the net. The way to calculate the number of parameters P is given by:

$$\mathrm{P} = (2 \times n + 1) + (m - 2) \times (n^2 + n) + (n \times 1 + 1)$$

Therefore, we define the general form of the neural network:

$$\mathcal{N} = \mathrm{A}_m(\sigma \mathrm{A}_{m-1}(...(\sigma \mathrm{A}_1(\vec{x}, \vec{t}))...)), \tag{7}$$

where $\vec{x}$ is the given space vector and $\vec{t}$ is the given time vector.

Figure 8 illustrates an exemplar neural network architecture utilized throughout the experiments outlined in section 4.1, specifically analyzing the 1-soliton case. It should be noted that, for this particular experiment, a relatively modest primary neural network structure was employed. The formal expression of this net is given by:

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

The total number of parameters in this net is 57.

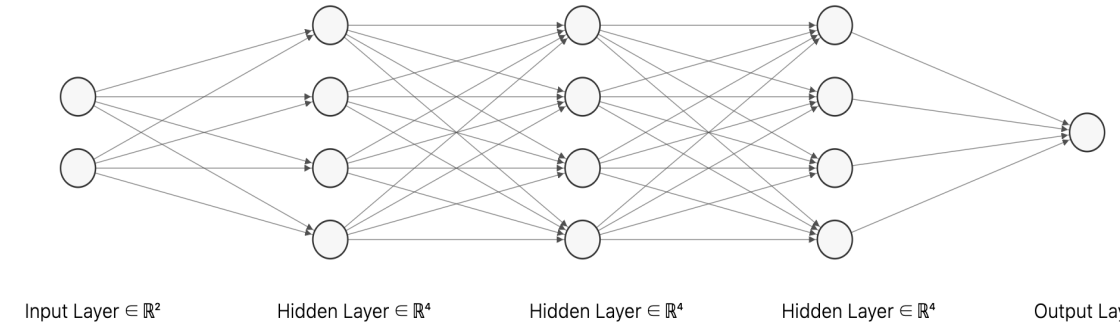


Figure 8: Neural Network Example Used

In practical experiments, varying sizes of neural networks will be implemented to solve distinct solutions for the Korteweg-de Vries equation 1. Additionally, the effects of utilizing large deep neural network structures on the experimental results will be extensively examined in section 4.4.

3.1.3 Empirical Risk Function

The crux of utilizing physics-informed neural networks (PINN) is to tailor deep neural networks to minimize empirical risk function for approximating the solution of a given partial differential equation (PDE). The residual items on points sourced from initial conditions, boundary conditions, and bulk comprise the empirical risk function (Raissi et al., 2017; Grossmann et al., 2023; Hu et al., 2022).

The vanilla PINN approach, as introduced by Raissi et al. (2017), uses the mean square error as the empirical risk of the approximator. For the approximator or the model, let us define it as $f_\theta(x,t)$ which is a combination of the neural networks, and θ represents the parameters in the neural net 7 and they are trainable. Besides, let $\mathcal{D}$ be the points set sampled from the domain of the PDE. The empirical loss $\hat{\mathcal{R}}$ of the general PDE equation 6 is given by:

$$\hat{\mathcal{R}} = \frac{1}{|\mathcal{D}_u|}\sum_{i=1}^{|\mathcal{D}_u|}|\frac{\partial f_\theta(x,t)}{\partial t} + \mathcal{L}[f_\theta(x,t);\lambda]|^2_{(x,t)\epsilon\mathcal{D}_u} + \frac{1}{|\mathcal{D}_b|}\sum_{i=1}^{|\mathcal{D}_b|}|f_\theta - g|^2_{(x,t)\epsilon\mathcal{D}_b} + \frac{1}{|\mathcal{D}_0|}\sum_{i=1}^{|\mathcal{D}_0|}|f_\theta - u_0|^2_{(x,t)\epsilon\mathcal{D}_0}, \tag{8}$$

where $\mathcal{D}_u$, $\mathcal{D}_b$, $\mathcal{D}_0$ denote the points set sampled from the bulk, the boundary condition and the initial condition (t=0) respectively.

The following section will outline the concept of population risk $\mathcal{R}$ in tandem with the empirical risk function 8 utilized within the study. Of note, Sirignano & Spiliopoulos (2018) developed an analogous methodology, employing the population risk function as a loss function. Apply the formula into the general PDE equation 6, we have:

$$\mathcal{R} = \|\frac{\partial f_\theta(x,t)}{\partial t} + \mathcal{L}[f_\theta(x,t);\lambda]\|^2_{2,\Omega\times[0,T],\nu_1} + \|f_\theta - g\|^2_{2,\partial\Omega\times[0,T],\nu_2} + \|f_\theta - u_0\|^2_{2,\Omega,\nu_3}, \tag{9}$$

where ν_y represents different positive probability densities.

The reason why they are the same is that the L2-norm in the population risk (equation 9) for a function is the integration on its domain, where $\mathcal{D}$ in empirical risk (equation 8) is sampled.

The integration process can be readily computed as the mean of the summation of corresponding values on each divided point when dividing a domain into multiple smaller sections.

To summarize, empirical loss function and population risk utilized within vanilla physics-informed neural networks have been previously outlined. The connections between these quantities has also been elucidated.

Moreover, in order to compare with the exact solution as a reference for evaluating the model's efficacy, it is necessary to introduce the fractional error function:

$$\text{Fraction Error} = \frac{\|f_\theta - u\|_2^2}{\|u\|_2^2},$$

where f_θ is the trained model and u is the exact solution.

3.1.4 Models with improvement

One of the greatest contributions of this work is that we developed the model which is used to approximate the solution of PDEs.

Current popular methods (Hu et al., 2022; Jagtap et al., 2020; Gurieva et al., 2022; Raissi et al., 2017) train pure neural networks capable of attaining true solutions across an entire interval. However, there may be a loss of accuracy inherent to these methods due to inefficiencies in leveraging known initial conditions and boundary conditions. In contrast, we opt to incorporate our model with prespecified boundary and initial conditions, occasionally resulting in superior performance. We name the pure net model as the vanilla model, while the model connecting the initial condition can be initial condition training-free, which means that we

don't need to train our net on the $t = 0$ situation. Similarly, the model connecting the boundary condition is boundary condition training-free.

To establish more lucid representation of our models, we have provided a general models f formula detailing the construction process below:

$$
\begin{aligned}
&f_{\text{vanilla}}(x,t) \coloneqq \mathcal{N}(x,t), \\
&f_{\text{boundary-included}}(x,t) \coloneqq \mathcal{N}(x,t) \cdot \frac{-a+x}{b-a} \cdot \frac{b-x}{b-a} + \frac{b-x}{b-a} \cdot g_a(a,t) + \frac{-a+x}{b-a} \cdot g_b(b,t), \\
&f_{\text{initial-included}}(x,t) \coloneqq \mathcal{N}(x,t) \cdot \frac{t^2}{t^2+q} + \frac{q}{t^2+q} \cdot u_0(x,0),
\end{aligned}
\tag{10}
$$

where $[a,b] = \Omega$ and q in $f_{\text{initial-included}}$ is a very small number. Besides, all $f_{\text{vanilla}}, f_{\text{boundary-included}}$ and $f_{\text{initial-included}}$ have instantiated the f_θ notation.

The boundary-included model incorporates known boundary conditions into the training process in addition to observational points, effectively rendering the boundary condition training-free. This approach can be particularly useful when boundary conditions play a critical role in determining system behaviour. Similarly, the initial-included model incorporates known initial conditions into the training process in addition to observational points, providing a train-free solution for accurate prediction of initial conditions. This approach is particularly valuable in scenarios where accurate prediction of initial conditions is key to predicting system evolution over time. Therefore, we can remove the corresponding terms in the whole empirical loss function:

$$
\begin{aligned}
\hat{\mathcal{R}}_{\text{vanilla}} &= \frac{1}{|\mathcal{D}_u|} \sum_{i=1}^{|\mathcal{D}_u|} |\frac{\partial f_\theta(x,t)}{\partial t} + \mathcal{L}[f_\theta(x,t);\lambda]|^2_{\mathcal{D}_u} + \frac{1}{|\mathcal{D}_0|} \sum_{i=1}^{|\mathcal{D}_0|} |f_\theta - u_0|^2_{\mathcal{D}_0} + \frac{1}{|\mathcal{D}_b|} \sum_{i=1}^{|\mathcal{D}_b|} |f_\theta - g|^2_{\mathcal{D}_b}, \\
\hat{\mathcal{R}}_{\text{boundary-included}} &= \frac{1}{|\mathcal{D}_u|} \sum_{i=1}^{|\mathcal{D}_u|} |\frac{\partial f_\theta(x,t)}{\partial t} + \mathcal{L}[f_\theta(x,t);\lambda]|^2_{\mathcal{D}_u} + \frac{1}{|\mathcal{D}_0|} \sum_{i=1}^{|\mathcal{D}_0|} |f_\theta - u_0|^2_{\mathcal{D}_0}, \\
\hat{\mathcal{R}}_{\text{initial-included}} &= \frac{1}{|\mathcal{D}_u|} \sum_{i=1}^{|\mathcal{D}_u|} |\frac{\partial f_\theta(x,t)}{\partial t} + \mathcal{L}[f_\theta(x,t);\lambda]|^2_{\mathcal{D}_u} + \frac{1}{|\mathcal{D}_b|} \sum_{i=1}^{|\mathcal{D}_b|} |f_\theta - g|^2_{\mathcal{D}_b},
\end{aligned}
\tag{11}
$$

However, the incorporation of prior knowledge into each model also introduces additional terms that cannot be trained, thereby increasing the lower bounds of our error. This disadvantage is particularly pronounced in the boundary-included model. In the initial-included model, the impact of these extra terms has been minimized by setting a small q. Further analysis of the performance of each model can be found in Section 5.

3.1.5 Training

The setup of the training procedure has been listed below:

1. Deep Learning Library: PyTorch
2. Server: UoM CSF (aka Danzek)
3. GPU: Nvidia V100

In all experiments involving various soliton number cases, extensive epochs have been utilized to facilitate visualization of performance changes. Despite robust high-performance computing support, these experiments demand significant time and resources. In an effort to ensure fair comparison and rigorous evaluation of the training process, each model is subjected to repetitions, allowing further examination of both variance and average error. This method enables a more objective measurement of the models' overall effectiveness.

The table below clearly recorded the time cost of each experiment.

Experiment	Total Time Cost(h)
1-Soliton with 57 parameters	4.5
1-Soliton with 541 parameters	7.6
1-Soliton with 1009 parameters	8.3
1-Soliton with 1981 parameters	9.25
2-Soliton with 417 parameters	13.1
2-Soliton with 2109 parameters	14.4
2-Soliton with 4137 parameters	18.9
2-Soliton with 10221 parameters	32.4
3-Soliton with 3500 parameters	21.3
"Double Descent" on 1-Soliton case	36.3

Table 1: Time cost for each experiment

3.2 Different size of DNN

By the analysis work in the coming sections 4.1, 4.2 and 4.3, it is obvious to find how the size of the neural network can significantly affect the performance. We will see how the empirical loss can reduce with more parameters but a contrary tendency on the fractional error in the 1-soliton case, while the same tendency on the error in the 2-soliton case. Scholars have extensively examined the relationship between the number of parameters requiring training and the amount of corresponding training data. Specifically, as explored by Pothuganti (2018) and Bashir et al. (2020), the ratio of the aforementioned quantities determines whether the model is under-fit or over-fit. The desired outcome is often to achieve a balance within this ratio in order to improve efficacy and performance. Besides, the model of Chat-GPT with billions parameters also inspired people that whether larger model can have better performance.

A recent area of interest in deep learning research is 'Double Descent' (depicted in Figure 9), as demonstrated by Nakkiran et al. (2019). This phenomenon highlights an intriguing pattern where, for a plot of test error against net size, the performance trends towards suboptimal before gradually improving once more.

Figure 9: Double Descent (Nakkiran et al., 2019)

Many people tried to explain this phenomenon and one interesting theory is the "Bias-Variance Trade-off" given by Belkin et al. (2019), while some people focus on how to optimize the "double descent". As introduced by Nakkiran et al. (2021), the optimal regularization on the weight decay is able to mitigate the "double descent".

We shall be noted that the phenomenon does not always exist in all cases and therefore we hope to research whether it will happen in PDE-solving problems. If so, whether the solution from Nakkiran et al. (2021) can mitigate it.

4 Results

4.1 Single soliton solution of KdV experiments

4.1.1 Models & Empirical Risk

Followed by the PDE equation 1, we will set the domain of this KdV equation to $x \in [0,1]$ and $t \in [0,1]$, and some experiments have been done to compare the special model 10 changes with the pure net model. Therefore, for the model equation 10, we set $a = -1$ and $b = 1$ for the $f_{\text{boundary-included}}(x,t)$ and for the $f_{\text{initial-included}}(x,t)$, the q used is $q = 10^{-9}$. The formulas of these models are given below when the 1 soliton solution of KdV has a space interval [0, 1] and time interval [0, 1]:

$$
\begin{aligned}
&f_{\text{vanilla}}(x,t) := \mathcal{N}(x,t),\\
&f_{\text{boundary-included}}(x,t) := \mathcal{N}(x,t)\cdot x\cdot(1-x) + x\cdot g(1,t) + (1-x)\cdot g(0,t),\\
&f_{\text{initial-included}}(x,t) := \mathcal{N}(x,t)\cdot\frac{t^2}{t^2+q} + \frac{q}{t^2+q}\cdot u_0(x,0)
\end{aligned}
$$

We can see one case that for the boundary-included model, when $x = 1$, the model will be exactly equal to the boundary condition function $g(1,x)$ and there is no need to train on this part. When $t = 0$, the initial included model will be equal to the initial condition u_0.

Besides, the empirical loss function of the KdV equation 1 corresponding to the risk equation 11 is given below:

$$
\begin{aligned}
\hat{\mathcal{R}}_{\text{vanilla}} &= \frac{1}{|\mathcal{D}_u|}\sum_{i=1}^{|\mathcal{D}_u|}\left|\frac{\partial f_\theta}{\partial t} + \frac{\partial^3 f_\theta}{\partial x^3} + 6 f_\theta\frac{\partial f_\theta}{\partial x}\right|^2_{\mathcal{D}_u} + \frac{1}{|\mathcal{D}_b|}\sum_{i=1}^{|\mathcal{D}_b|}|f_\theta - g|^2_{\mathcal{D}_b} + \frac{1}{|\mathcal{D}_0|}\sum_{i=1}^{|\mathcal{D}_0|}|f_\theta - u_0|^2_{\mathcal{D}_0},\\
\hat{\mathcal{R}}_{\text{boundary-included}} &= \frac{1}{|\mathcal{D}_u|}\sum_{i=1}^{|\mathcal{D}_u|}\left|\frac{\partial f_\theta}{\partial t} + \frac{\partial^3 f_\theta}{\partial x^3} + 6 f_\theta\frac{\partial f_\theta}{\partial x}\right|^2_{\mathcal{D}_u} + \frac{1}{|\mathcal{D}_0|}\sum_{i=1}^{|\mathcal{D}_0|}|f_\theta - u_0|^2_{\mathcal{D}_0},\\
\hat{\mathcal{R}}_{\text{initial-included}} &= \frac{1}{|\mathcal{D}_u|}\sum_{i=1}^{|\mathcal{D}_u|}\left|\frac{\partial f_\theta}{\partial t} + \frac{\partial^3 f_\theta}{\partial x^3} + 6 f_\theta\frac{\partial f_\theta}{\partial x}\right|^2_{\mathcal{D}_u} + \frac{1}{|\mathcal{D}_b|}\sum_{i=1}^{|\mathcal{D}_b|}|f_\theta - g|^2_{\mathcal{D}_b},
\end{aligned}
\tag{12}
$$

In this experiment, as the boundary condition has been included in the model, the penalty term on the boundary can therefore be removed. For the initial-included model, there is no penalty term. Furthermore, as we are using the same PDE in all multi-soliton solution experiments, this empirical risk equation 12 is always the same and therefore readers can come back here to recap the empirical risk functions used in the KdV experiment.

4.1.2 Reference Solution

Now we shall define the initial condition and the boundary condition for the single-soliton case. For the single-soliton case, the conditions followed the same configuration as Drazin & Johnson (1989) did, where the initial condition is $u_0(x) = N(N+1)\,\text{sech}^2(x)$:

$$
\begin{aligned}
u_0(x) &= 2\,\text{sech}^2(x)\\
g(x,t) &= \begin{cases} u(0,t), x = 0\\ u(1,t), x = 1, \end{cases}
\end{aligned}
$$

where u denotes the exact single soliton solution we got from 2 in the former section. This may look strange as we are trying to derive a solution we already used in the boundary condition, but in fact, the space domain of the KdV equation is usually defined as infinite (Drazin & Johnson, 1989), which means that we have to set the boundary condition manually to apply in the model or loss.

In the limited domain, the training dataset for the three PINN models 10 comprises 500 residual points (bulk), 250 boundary points (with 125 each from $x = 0$ and $x = 1$), and 250 initial points ($t = 0$). The testing dataset for these models consists of 10,201 points uniformly distributed within the domain. The neural

network's depth is fixed at four layers, and throughout the experiments, we will investigate the impact of under-parameterized and over-parameterized networks on the results by adjusting the network's width.

The activation function employed is the "Sigmoid" function. To initialize the parameters of the neural net, Xavier normalization is used. For optimization purposes, the Adam optimizer (Kingma & Ba, 2017) with a learning rate of 10^{-4} is utilized, and no regularization is applied during the experimentation process.

To comprehensively evaluate and showcase the generalization capabilities of the various models, it is of paramount importance that the size of the testing point set substantially exceeds that of the training point set. Furthermore, to establish fairness and rigour across these experimental comparisons, all models should be subjected to a series of multiple training iterations (in this particular study, we implemented 3 repetitions). This deliberate methodological approach allows for a meticulous examination of not only the variance but also the precise average values in the testing scenarios, fostering robust and insightful conclusions about model performance.

4.1.3 Results

For a fair comparison, we keep the same training procedure, i.e., training epochs, learning rate, model structure, etc. We train each model for 60,000 epochs, and the tabular below the table 2 lists the empirical risk and the fractional error w.r.t true solution of each model with different P / N ratios at the 60,000-th epoch:

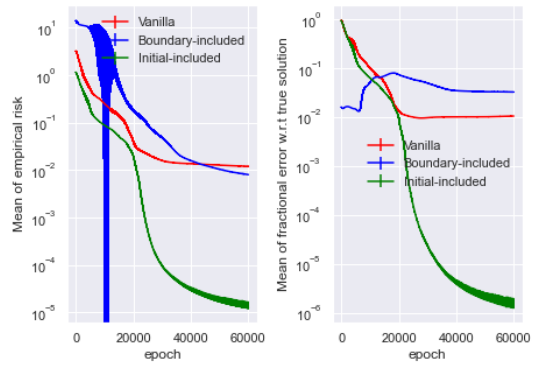
1-Soliton Solution of KdV (Epoch =60000, Repeat=3, Test Size=10201)					
Parameter Number(P)	Sample Size(N)	P / N ≈	Model	Empirical Risk	Fractional Error
57	Bulk: 500 t=0: 250 x=0: 125 x=1: 125	0.05	Vanilla	1.20×10^{-2}	1.05×10^{-2}
			Boundary-included	8.04×10^{-3}	3.24×10^{-2}
			Initial-included	1.24×10^{-5}	1.39×10^{-6}
541		0.5	Vanilla	4.06×10^{-5}	2.50×10^{-4}
			Boundary-included	7.27×10^{-4}	2.29×10^{-2}
			Initial-included	4.51×10^{-6}	6.22×10^{-6}
1009		1	Vanilla	1.58×10^{-4}	8.57×10^{-3}
			Boundary-included	1.77×10^{-4}	8.99×10^{-3}
			Initial-included	3.72×10^{-6}	6.60×10^{-5}
1981		2	Vanilla	1.29×10^{-5}	3.00×10^{-4}
			Boundary-included	2.89×10^{-5}	1.53×10^{-3}
			Initial-included	3.49×10^{-6}	1.63×10^{-5}

Table 2: Table of the Results of the 1-Soliton Solution from Different Models and Different P / N Ratio

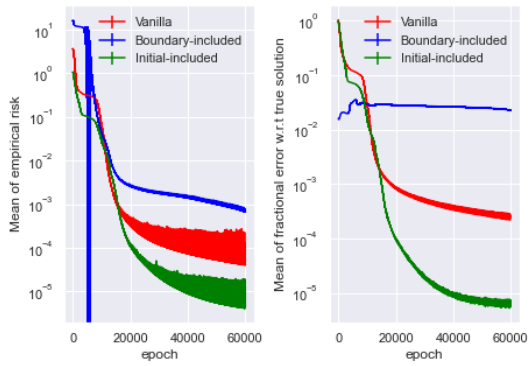
This experiment trained 12 different neural networks. The results presented in red font indicate the lowest achievable fractional error: 1.39×10^{-6}, and intriguingly, the value was obtained from the initial-included model with the lowest P/N ratio. Additionally, it is worth noting that for both high and low P/N ratios, the model inclusive of initial conditions consistently performs best, with fractional error found to be the lowest in all cases. However, for using a larger scale neural network, the performance is getting worse for the initial-included model.

The boundary-included model also exhibited a distinct but opposite trend, with an increase in neural net size resulting in a noticeable decrease in fractional error. In contrast, the vanilla model yielded a unique pattern where fractional error initially decreased upon increasing the neural net size, followed by a subsequent increase and then a final decrease in error as the net size grew larger. This is unstable and therefore hard to conclude anything about how neural net size can affect the vanilla mode.

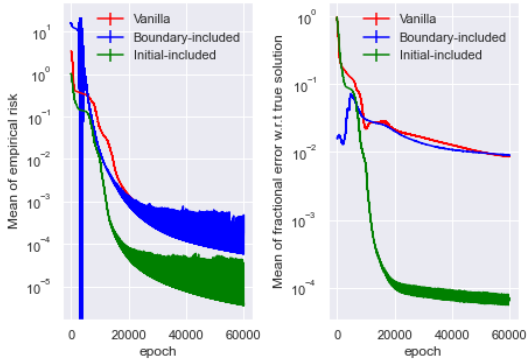
Visualizing the trends in results across epochs, we found that a table alone was insufficient in capturing the distinct changes observed. Therefore, plots were generated to more clearly represent these shifts and demonstrate the evolving trends across time. Figure 10 presents the error bars representing both the training loss and the fractional error with respect to the true solution, derived from the statistical outcomes of 3 repetitions.



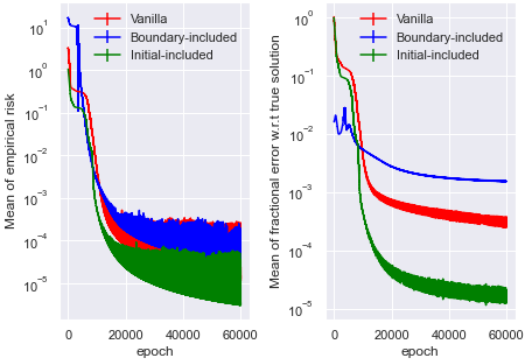
(a) Results for the 54 parameters net



(b) Results for the 541 parameters net



(c) Results for the 1009 parameters net



(d) Results for the 1981 parameters net

Figure 10: Computational Results of 1-Soliton Solution KdV with Different Size Neural Nets

To generate the error bar plots, we calculated both variance and mean results for each model. The coloured regions on the plot, which are not slim curves, indicate times when variance was high. This approach enables clearer visualization of both the trends observed across epochs and the relative performance of each model

By evaluating the variance, the plots effectively illustrate the average results across multiple iterations, thereby providing a comprehensive understanding of the model's performance.

The four plots distinctly reveal each model's performance when subjected to increased training epochs. A consistent feature observed across all plots is that when the number of epochs remains below approximately 5,000, the boundary-included model (blue) exhibits the most optimal performance. However, with a substantial increase in epochs, the initial-included model (green) ultimately surpasses the other two models. This seems a trade-off of accuracy and time.

Additionally, we observed that the initial-included model's performance improved rapidly before stabilizing around 20,000 epochs. Despite the model's higher variance, prolonged training times yielded significant performance improvements. Furthermore, as parameter size increased, fractional error decreased at a faster rate, indicating the importance of carefully optimizing neural network architecture and design to yield the best possible results.

Analysis of the performance of the vanilla model (red) in Figure 10 highlights that its efficacy can be significantly impacted by neural net size. Specifically, larger models resulted in a lower fractional error and faster rates of decrease. The four plots presented in this study demonstrate that the vanilla model (red

curve) exhibits similar patterns to the initial-included model (green curve). Given the model function used, it is reasonable to conclude that by providing prior knowledge of initial conditions, it becomes possible to inspire improved models with better fits while utilizing identical epoch times and neural networks. These findings suggest that leveraging available information can enhance model performance when confronted with complex PDEs.

Interestingly, results from extended training for the boundary-included model also revealed that while possessing prior knowledge of boundary conditions enabled stronger performance initially, this advantage became disrupted over time. These findings emphasize the importance of careful consideration when integrating prior knowledge into PINN models, particularly as training progresses. Further development of the boundary-included model may involve adding weights to prior knowledge terms, allowing for more nuanced integration of available information. Despite the high cost associated with training, the trade-off phenomenon observed in our experiments underscores the continued value of the boundary-included model.

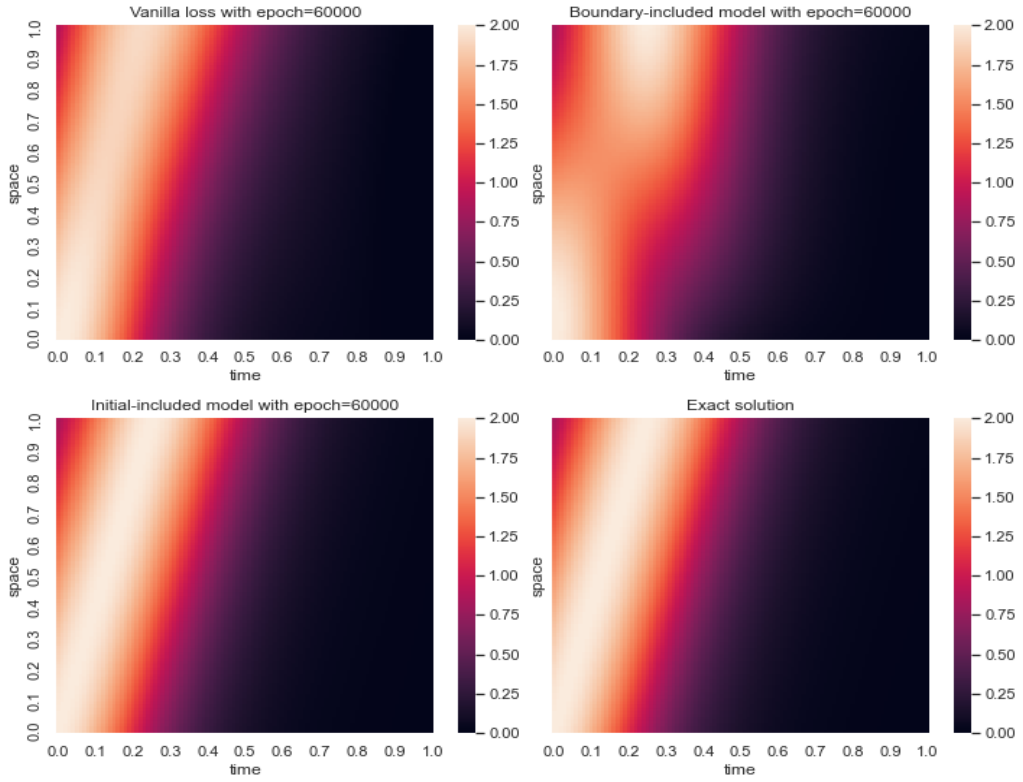


Figure 11: Heat map of the 1-soliton solution using 3 models with 57 parameters and the exact solution: Left top: Vanilla model; Right top: Boundary-included model; Left bottom: Initial-included model; Right bottom: Exact solution

In summary, Figure 11 provides a visual representation of the three PINNs models' ability to simulate the KdV equation for a single soliton solution. The heat map, with the X-axis representing time and the Y-axis denoting space, displays a prominent peak with a height of approximately 2. As time progresses, this peak can be observed travelling from $x = 0$ to $x = 1$, effectively capturing the dynamics of the underlying soliton solution. This heatmap 11 also directly shows the difference in the performance among these three models.

Consequently, our findings suggest the existence of a trade-off between accuracy and time: under constraints on computational resources or time, the boundary-included model exhibits superior performance, but finally, the initial-included model yields enhanced accuracy. Besides, we analyzed how the performance of the vanilla model can be developed by integrating the prior knowledge of the initial condition and found the reason that it has a similar but just worse performance when compared with the initial-included model for using the same epoch or neural net. This observable relationship underscores the importance of considering computational demands when selecting an appropriate model for varying circumstances.

Next, we will focus on the KdV equation with 2-solitons solution case, where the PDE is more complex to approximate because of the interaction region of the 2 solitons.

4.2 2-soliton solution of KdV experiments

4.2.1 Models & Empirical Risk

Followed by the PDE equation 1 with the 2-soliton solution 3, the domain in this experiment is larger: $x \in [-5, 5]$ and $t \in [-1, 1]$. Besides, more points are sampled to match the larger domain: 4000 residual points (bulk), 2000 initial points ($t = 0$) and 2000 boundary points (1000 from $x = -5$ and 1000 from $x = 5$). Therefore the size of the testing points is also increased: 60501 (301×201) points uniformly distributed within the domain. This setup is special as we are considering the points from a negative time space.

Once more, to proceed with the models employed for experiments involving the 2-soliton solution, the equation 10 from the section 3.1.4 needs to be recapped. By setting $a = -5, b = 5$ for the $f_{\text{boundary-included}}$ and setting $q = 10^{-9}$ for the $f_{\text{initial-included}}$, the models for this case can be displayed.

We have now ascertained that for points sampled from the boundaries, the boundary-included model can accurately yield the known boundary conditions, while for points drawn from the initial states, the initial-included model can generate $u_0(x)$ without any error. Consequently, the relevant penalty terms may be eliminated from the loss function of the vanilla PINN, streamline its formulation, without compromising the integrity of the results. The empirical risk function has been claimed in the equation 12.

4.2.2 Reference Solution

In the 2-soliton solution experiments, in order to follow the same PDE setup as Drazin & Johnson (1989), the exact solution we used is in equation 3:

$$u(x,t) = 2(\log f)_{xx},$$

where

$$f = 1 + e^{\eta_1} + e^{\eta_2} + e^{\eta_1+\eta_2+A_{12}}$$
$$\eta_i = k_i x - k_i^3 t + \eta_i^{(0)}$$

and

$$e^{A_{ij}} = \left(\frac{k_i - k_j}{k_i + k_j}\right)^2.$$

For the same setup, $k_1 = -2, k_2 = 4, \eta_1^{(0)} = \log\frac{1}{3}$ and $\eta_2^{(0)} = \log\frac{1}{3}$.

These parameters play a crucial role in obtaining various forms of solitons for the KdV equation. While they do not determine the correctness of the solution, they significantly influence the shape and characteristics of the resulting solitons, thereby contributing to the diverse manifestations of the equation's solutions. In this group of parameters, they guaranteed that the two solitons will be overlapped when $t = 0$, where there is only one pulse when $t = 0$ so that the solution can match the initial condition given below.

Consequently, for the known conditions of this PDE, we have an initial condition which is demonstrated by Drazin & Johnson (1989) with the same pattern, but still owns a non-exact boundary condition which we need the help from the exact solution:

$$u_0(x) = 6\,\mathrm{sech}^2(x), t = 0$$

$$g(x,t) = \begin{cases} u(-5,t), x = -5 \\ u(5,t), x = 5, \end{cases}$$

4.2.3 Results

As in the previous experiments, we maintained a fixed depth of 4 layers for the neural network while continuing to investigate large-scale networks by varying their width. The activation function utilized remains the "Sigmoid" function, and Xavier normalization is employed for initializing the neural network parameters. The Adam optimizer with a learning rate of 10^{-4} continues to be our selection, and no regularization is implemented during this stage of experimentation.

We once again conducted the training process four times to measure both the variance and average performance of each model, resulting in a tabular 3 representation and figure 12 with a similar format as before. To reinforce the persuasiveness of the model's generalization abilities, we maintained the approach of utilizing a larger testing point set for all subsequent evaluations.

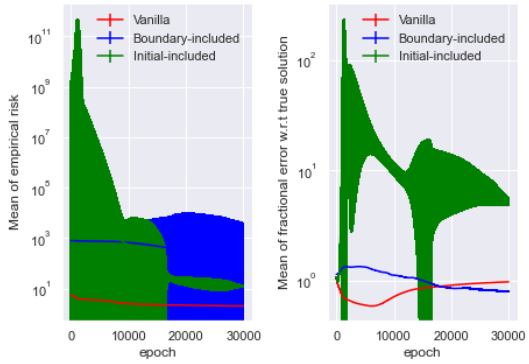
2-Soliton Solution of KdV (Epoch =30000, Repeat=3, Test Size=60000)					
Parameter Number(P)	Sample Size(N)	P / N ≈	Model	Empirical Risk	Fractional Error
417	Bulk: 2000 t=0: 1000 x=-5: 500 x=5: 500	0.1	Vanilla	2.05	9.74×10^{-1}
			Boundary-included	$1.44e \times 10^{2}$	7.98×10^{-1}
			Initial-included	$1.25e \times 10^{1}$	5.16
2109		0.5	Vanilla	1.35	1.08
			Boundary-included	9.70×10^{11}	1.29
			Initial-included	8.04×10^{-2}	6.56×10^{-1}
4137		1	Vanilla	1.54	1.11
			Boundary-included	2.71×10^{1}	1.36
			Initial-included	5.84×10^{-2}	6.84×10^{-1}
10221		2.5	Vanilla	1.65	1.12
			Boundary-included	2.76×10^{1}	1.50
			Initial-included	1.21	8.84×10^{-1}

Table 3: Table of the Results of the 2-Soliton Solution from Different Models and Different P / N Ratio

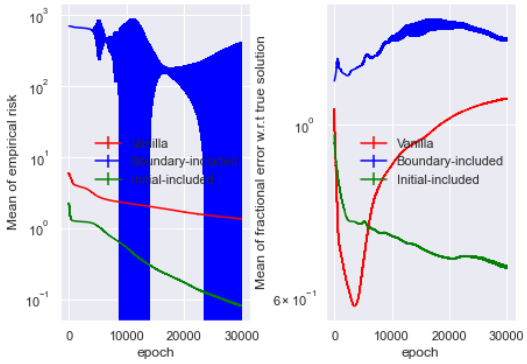
This experiment trained 12 different neural networks. In the table 3, the figure highlighted in red represents 6.56×10^{-1}, obtained from the initial-included model. This result is consistent with those acquired from single-soliton experiments after extensive epochs of training. Additional insights were gained from Table 3, which revealed that for the initial-included model, larger neural nets resulted in improved performance, with the smallest net producing inferior results. Conversely, both the vanilla and boundary-included models demonstrated the opposite trend, exhibiting worse performance as additional parameters become available, even though the empirical risk is reduced. This seems like the initial-included model is better at leveraging over-parameterization.

Moreover, it is worth noting that both empirical risk and fractional error were significantly higher than the results observed from the 1-soliton experiment, underscoring the challenges encountered when fitting complex PDEs using PINN models. These findings highlight the need for careful consideration of model structure and techniques to optimize the performance of PINN models when dealing with challenging and complex problems.

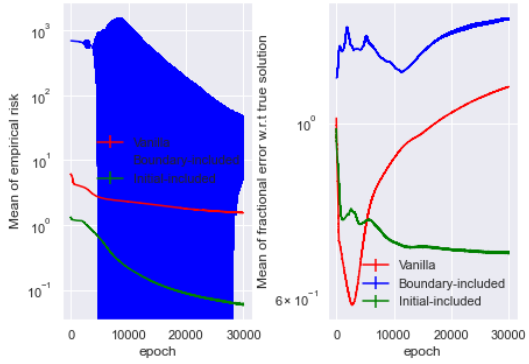
Equally, to visualize how the risk and the error change during the whole training process, the figure 12 is given below.



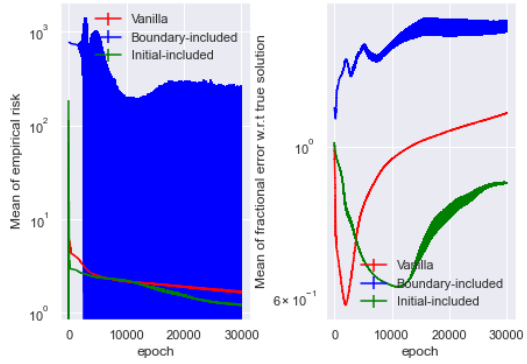
(a) Results for the 417 parameters net



(b) Results for the 2109 parameters net



(c) Results for the 4137 parameters net



(d) Results for the 10221 parameters net

Figure 12: Computational Results of 2-Soliton Solution KdV with Different Size Neural Nets

Same as the figure 10, the figure 12 shows the error bar of the results and the variance in these plots are much higher than we got before.

Despite the trade-off phenomenon remaining visible, our experiments revealed that the vanilla model has emerged as the representative model of efficiency, displaying superior performance for all four plots when having a limited source to train the model.

However, in the initial-included model, after prolonged training times, it remains the best model for all but the smallest neural net size in Figure 12(a). Here the subplot 12(a), the initial-included model exhibited very high variance during training and the boundary-included model dominated the vanilla model at around the 20,000th epoch.

It is interesting that all figures show that the fractional of the vanilla model first reduces and then increases rapidly after about 5000 epochs. The fractional error at the last epoch is usually worse than the error at the first epoch, even though the empirical risk is still decreasing. This may come from the over-fitting after lots of training and sampling more points on the domain probably solves the problem. This is meaningful as these four plots indicate that the lowest error the vanilla model can reach is lower than the best the initial-included model can be. Besides, we saw from the figure 12(d) than when there are 10221 parameters in the model, the initial-included model (green) also has the "U"-shape for the fractional error during the training.

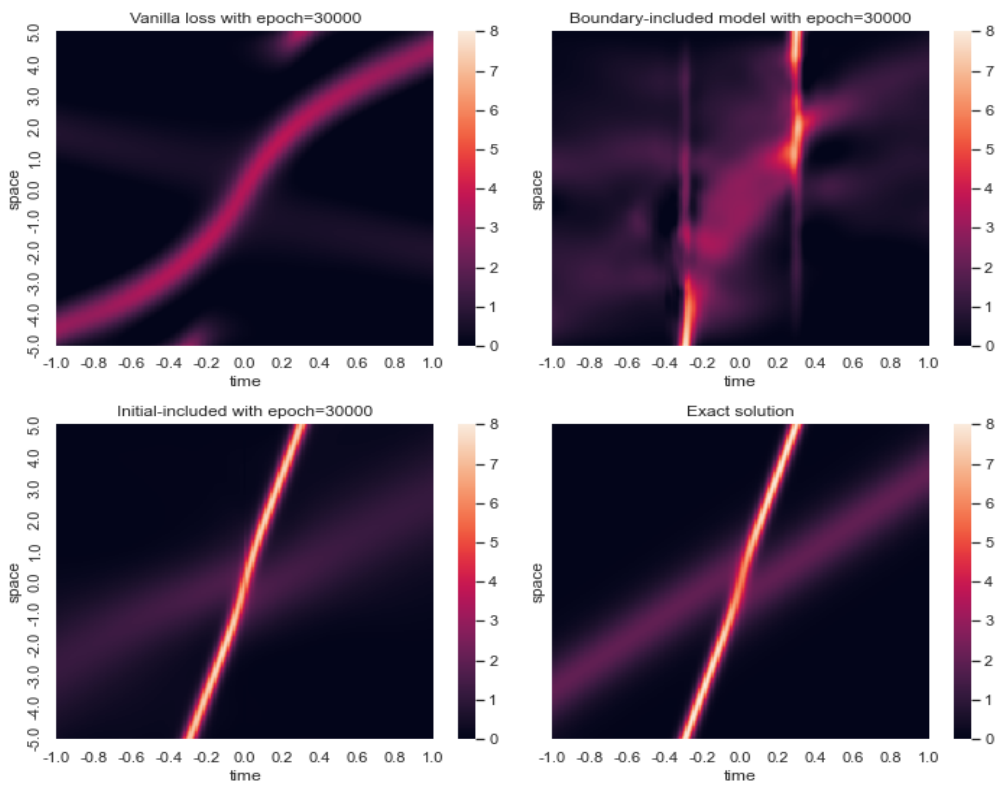


Figure 13: Heat map of the 2-soliton solution using 3 models with 2109 parameters and the exact solution: Left top: Vanilla model; Right top: Boundary-included model; Left bottom: Initial-included model; Right bottom: Exact solution

In summary, our analysis of the double-soliton experiment is encapsulated in Figure 13, which visualizes the movement of two solitons within the spatial domain over time. The heatmaps for the three PINN models exhibit significant differences compared to those observed in heatmap 11. Notably, the initial-included model performed the best, with prior knowledge of the initial condition contributing to improved fitting of the intersection region of the two solitons. Conversely, the boundary-included model, despite exhibiting a vertical line indicative of prior knowledge of boundary conditions, did not perform better. Finally, the vanilla model exhibited poor performance and failed to fit the PDE effectively.

Overall, our analysis of the 2-soliton case of KdV illustrates a classic example where vanilla PINNs models can be improved. Results from our experiments clearly demonstrate that integrating prior knowledge of initial conditions into the model yields significant performance improvements, highlighting the importance of strategic selection and utilization of available information when dealing with complex PDEs.

4.3 3-soliton solution of KdV experiments

4.3.1 Models & Empirical Risk

For the 3-soliton solution case, the configurations used are more different. From the former work done by Hu et al. (2022), we have a proper goal on the fractional error which is 6.89×10^{-1} for 3-soliton solution KdV and the aim is to try to beat the results from Hu et al. (2022) by using less number of parameters with the improved model. Besides, to make the task more challenging, the domain used is $x \in [-4, 4]$ and

$t \in [-0.5, 0.5]$. By using the same configuration as the target did, this is how the points are sampled from the domain: 18000 residual points from the bulk, 914 points from $t = 0$, and 914 points from the boundary(457 from $x = -4$ and $x = 4$). The testing set contains 102400 points uniformly distributed within the domain. Negative time-space was employed to facilitate clearer visualization of the soliton intersection region.

The models used in this case correspond to the model equation 10 set $a = -4, b = 4$ in the $f_{\text{boundary-included}}$ and set $q = 10^{-4}$ in the $f_{\text{initial-included}}$. And the empirical risk functions is still the same as defined in the equation 12.

4.3.2 Reference Solution

The 3-soliton exact solution used is the reference solution for calculating the fractional error. The solution is derived according to the book from Drazin & Johnson (1989) as before, which is the equation 5. Besides, to make sure that the initial condition is consistent, matching the initial condition to the equation 4, the 6 values of the parameters in the solution equation 5 are $k_1 = 2, k_2 = 4, k_3 = -6, \eta_1^0 = \ln\frac{3}{2}, \eta_2^0 = \ln\frac{3}{5}, \eta_3^0 = \ln\frac{1}{10}$.

The proof has been appended in Appendix B, and they are important to decide the shape and the movement of the solitons. By using this group of values, the three solitons will be fully overlapped when $t = 0$, where only one peak is visible.

The initial condition and the boundary condition applied in this experiment are:

$$u_0(x) = 12\,\text{sech}^2(x), t = 0$$

$$g(x,t) = \begin{cases} u(-4,t), x = -4 \\ u(4,t), x = 4, \end{cases}$$

4.3.3 Results

In contrast to the neural network used in Hu et al. (2022) experiment, which utilized approximately 4,000 parameters, one of our goals was to achieve superior performance using fewer parameters. To this end, we employed a neural architecture with a depth of 5 and a width of 32, resulting in a total of 3,500 trainable parameters - 500 fewer than in the previous study. The epoch number used by Hu et al. (2022) is 5,000 and the target fractional error is the 5,000-th result. However, in our experiment, we will use a larger epoch number but still record the 5,000-th results from each model for comparison.

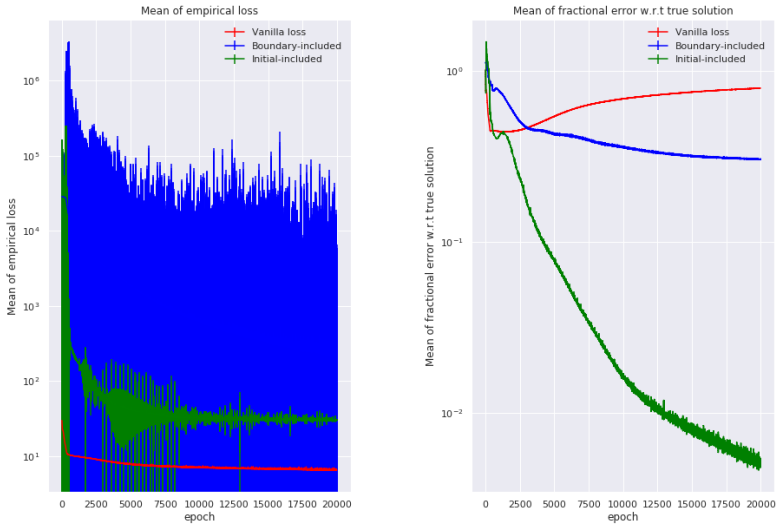
Additionally, several modifications were made to network configuration and optimization techniques in comparison to the study by Hu et al. (2022). Activation functions were changed to "tanh", as introduced in Section 2.2.3, while Xavier normalization was still employed for initialization of neural net parameters. The Adam optimizer was utilized for training, with a larger learning rate of 10^{-3} chosen. Finally, no regularization was implemented during experimentation. To obtain more robust and accurate results, all training processes were repeated three times to measure variance and average performance. Notably, our analysis of the 3-soliton case highlights the significant improvements possible through the utilization of the initial-included model. However, this experiment will not be repeated by varying the size of the neural nets and we only focus on one fixed size. Overall, 3 different neural networks are trained.

3-Soliton Solution of KdV (Epoch =5000, Repeat=3, Test Size=102400)			
Model	Sample Size(N)	Parameter Number	Fractional Error
Hu et al. (2022)	Bulk:18000 t=0: 914 x=-4: 457 x=4: 457	4000	6.9×10^{-1}
Vanilla Model		3500	6.0×10^{-1}
Boundary-Included Model			4.1×10^{-1}
Initial-Included Model			4.3×10^{-2}

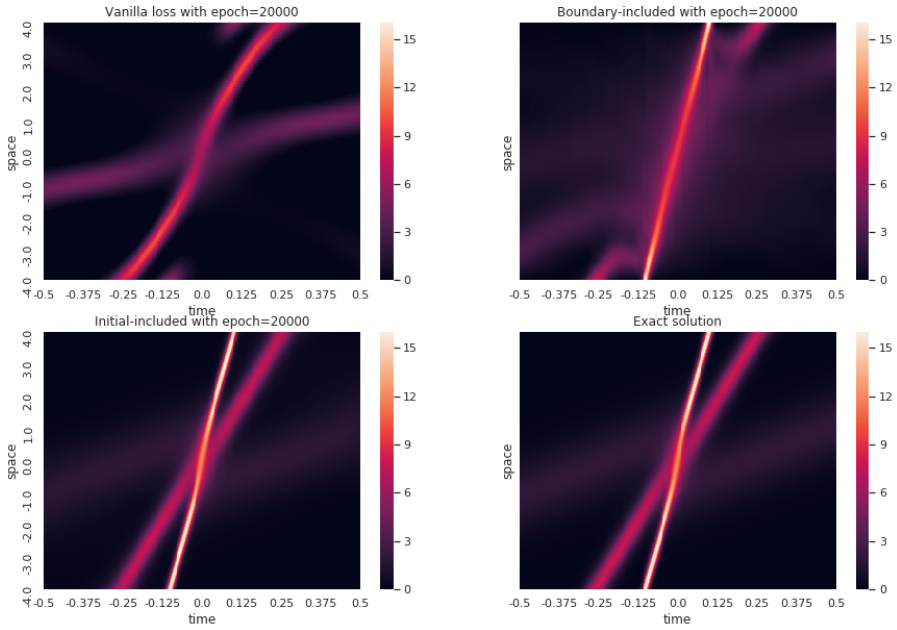
Table 4: Table of the Results of the 3-Soliton Solution from Different Models for the Comparison with the Work by Hu et al. (2022)

The fractional errors with respect to the exact solution at the 5000-th epoch are recorded in Table 4. Our improved models demonstrated superior performance compared to the results presented in Hu et al. (2022),

even when using fewer neural network parameters. Moreover, given that the fractional error of our vanilla model experiment is similar to that reported in Hu et al. (2022), we can reasonably infer that our experiments were valid. Specifically, the initial-included model exhibited a fractional error nearly ten times lower than that of the vanilla model, and this is a giant growth.



(a) Error bar for the 3-soliton models



(b) Heatmap for the 3-soliton models

Figure 14: Results of 3-Soliton Solution KdV

Figure 14, and specifically subplot 14(a), provides clear visualization of the performance differences between the three models when subjected to extended training epochs. Notably, our experiments revealed that the initial-included model significantly outperformed the vanilla model, while the boundary-included model also exhibited superior performance after approximately 2,000 epochs. Our experiments have demonstrated that the initial-included model, developed through careful integration of prior knowledge regarding the initial condition, can yield significant performance improvements. Additionally, we observed that the boundary-included model is also capable of outperforming the vanilla model when subjected to training epochs greater than 2,000.

An additional observation from subplot 14(a) is that even when subjected to 20,000 epochs, the decreasing gradient of the initial-included model (green) remains significant. This observation suggests that there may still be untapped potential for the initial-included model to further optimize its ability to fit the PDE. Figure 14 subplot 14(b) provides a visual representation of the performance of the three models at the 20,000th epoch. A direct comparison reveals that the initial-included model exhibits the closest approximation to the exact solution. Conversely, the vanilla model failed to effectively fit two of the solitons and exhibited confusion in the intersection region.

The 3-soliton solution KdV case provides further evidence of how vanilla PINN models can struggle to effectively fit complex PDEs, particularly in cases involving multiple solitons or intersection regions. However, our experiments highlight the potential for significant performance improvements through the integration of prior knowledge with neural network structures, as demonstrated by both the initial-included model and the boundary-included model. And the most important thing is that we achieved better results using the initial-included model with fewer neural net parameters than Hu et al. (2022) did.

4.4 Experiments on "Double Descent"

The experiments in this section mainly focused on the 1-soliton solution with the same configuration as we did in section 4.1 to verify if the "Double Descent" phenomenon exists in the vanilla PINNs PDE-solving problem. Besides, instead of regularising on the weight decay as what Nakkiran et al. (2021) did for mitigating, we applied a weight λ on the penalty terms in the loss function 8 to show how the λ weight can affect the performance when the size of the neural net is getting larger. Furthermore, to set up a comparison group, the boundary-included model is also considered in this experiment.

Therefore, by adding the factor λ in the loss function, we have:

$$\hat{\mathcal{R}}_{\text{vanilla}} = \frac{1}{|\mathcal{D}_u|}\sum_{i=1}^{|\mathcal{D}_u|}|\frac{\partial f_\theta}{\partial t} + \frac{\partial^3 f_\theta}{\partial x^3} + 6f_\theta\frac{\partial f_\theta}{\partial x}|^2_{\mathcal{D}_u} + \frac{\lambda}{|\mathcal{D}_b|}\sum_{i=1}^{|\mathcal{D}_b|}|f_\theta - g|^2_{\mathcal{D}_b} + \frac{\lambda}{|\mathcal{D}_0|}\sum_{i=1}^{|\mathcal{D}_0|}|f_\theta - u_0|^2_{\mathcal{D}_0}$$

for the vanilla model and:

$$\hat{\mathcal{R}}_{\text{boundary-included}} = \frac{1}{|\mathcal{D}_u|}\sum_{i=1}^{|\mathcal{D}_u|}|\frac{\partial f_\theta}{\partial t} + \frac{\partial^3 f_\theta}{\partial x^3} + 6f_\theta\frac{\partial f_\theta}{\partial x}|^2_{\mathcal{D}_u} + \frac{\lambda}{|\mathcal{D}_0|}\sum_{i=1}^{|\mathcal{D}_0|}|f_\theta - u_0|^2_{\mathcal{D}_0}$$

for the boundary-included model.

The maximum-size neural network employed in our experiments consisted of approximately 2,000 parameters, while the total number of training points was 800 (400 from the bulk, and 200 from each boundary and initial condition). This corresponds to a maximum ratio of 2.5 between the number of parameters and the number of training points. Training for this largest network required only 1,500 epochs, which was deemed sufficient after extensive verification. Additionally, we tested regularization parameter λ values ranging from a small number to over 1, including 0.0005, 0.003, 0.01, 0.95, 1, and 1.05. The total number of trained neural networks in this experiment is 252, 126 from each model, and 21 from each λ setting.

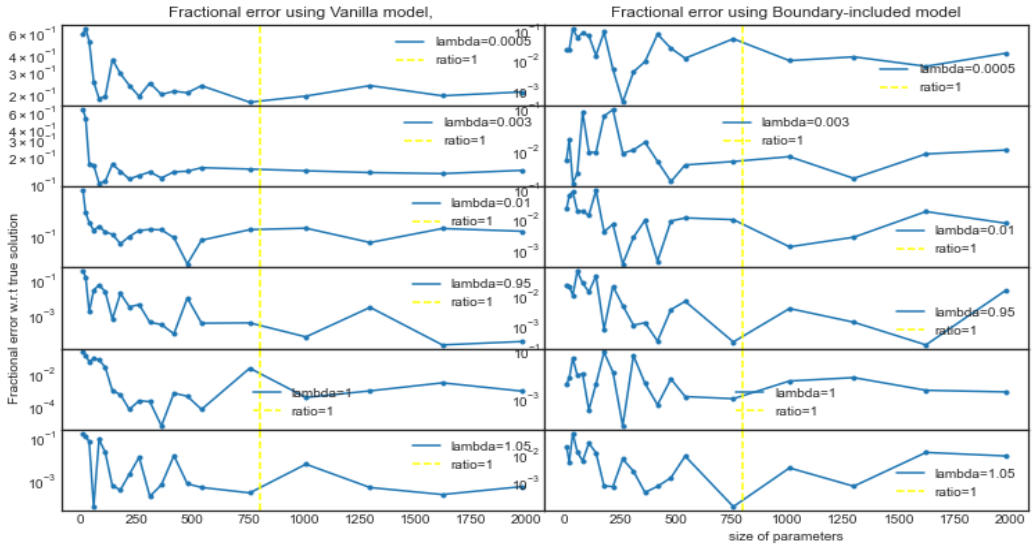


Figure 15: Left: Fractional error for the vanilla model; Right: Fractional error for the boundary-included model

In Figure 15, it is hard to define where the second descent comes from no matter what the λ is. Even though is the vanilla model side with $\lambda = 1$, we can see there is one peak around the yellow line, which is the place where the double descent highest peak should be (Nakkiran et al., 2021), it is still hard to explain the double descent is there as there is only one point supporting the peak. Besides, according to Nakkiran et al. (2019), after the second descent, the fractional error is supposed to get lower, but figure 15 does not have the features.

While it may not be appropriate to conclusively state that "Double Descent" effects exist in PINN models no matter if prior knowledge is integrated, our experiments do suggest that increasing neural network size does not always lead to reductions in fractional error.

5 Reflection & Conclusion

In this section, we will evaluate and reflect on the project as a whole, listing some potential works that can expand the current work. Finally, we will also offer a conclusion of the entire study on solving KdV equation using PINNs.

5.1 Planning

One of the main challenges with this study is about how to design the new models 10 ingeniously. As a combination with the neural network and the prior knowledge, we hope that the solution can be fully replaced by the known conditions when at $t = 0$ or boundaries. And we do not wish that this prior knowledge can disturb the models training as these extra terms are not trainable. From the boundary included model equation in equation 10, we can see that these prior knowledge is still affecting model when x is not at the boundaries. Some may think that this can inspire the model training but it cannot be rejected that these non-trainable terms can also disturb the model even when they are not needed. The first version of the initial-included model also has this problem, but by introduing the tiny parameter q, the effect from the extra term can be minimized when $t \neq 0$. Therefore, there is still a zone to give a better design on the boundary-included model.

Future work can build upon our experiments involving the KdV equation 1, in particular, exploring solutions involving more solitons. Our results demonstrated that the initial-included model outperformed other models when the PDE was more complex, but additional exploration is needed to extend these findings to more soliton solutions (say, 5-solitons). Given the importance of the KdV equation and its widespread use in dynamic systems, demonstrating the ability of PINNs to solve it efficiently and accurately with more solitons would be a valuable contribution to the field of scientific machine learning.

Therefore, by reflecting the results of this report, there are still many potential works on solving multi-soliton solution KdV equations using PINNs.

5.2 Conclusion

This report represents an exploration of the application of scientific machine learning to fundamental subjects. By combining deep learning techniques with physics-driven models, we uncovered exciting possibilities for future research in this field. We learned that Physics-Informed Neural Networks (PINNs) are a promising technique in this area, and thus our work focused on applying PINNs to solve the Korteweg-De Vries equation - a special PDE with wide applications in wave motion and soliton dynamics.

By integrating prior knowledge such as initial and boundary conditions into vanilla PINNs, we conducted experiments on the KdV equation for cases involving single, double, and triple soliton solutions. Our results indicated that for the single-soliton solution case, the initial-included model was the best choice, with a trade-off between time and accuracy when considering the boundary-included model if the training was stopped early. For the double-soliton solution case, the initial-included model remained the best for fitting the KdV equation, but we observed another trade-off wherein the vanilla model performed better when training time was limited. In the most complex example - the triple-soliton solution case - we found that the initial-included model performed best regardless of the training time.

In addition to exploring the PINN models for solving the KdV equation, we also investigated the "Double Descent" phenomenon in larger scale neural networks. After conducting experiments on the single-soliton solution case of the KdV equation and varying the size of neural network parameters, we found no evidence of the "Double Descent" phenomenon occurring in PDE-solving problems using deep learning.

In summary, this study has demonstrated that integrating prior knowledge such as initial and boundary conditions into vanilla PINNs models can significantly improve performance and accuracy when solving complex PDEs. This improvement offers promising opportunities for future research in this area, inspiring further studies on how to effectively incorporate known conditions into PINNs to enhance their performance. Overall, our work highlights the potential of PINNs for solving PDEs using deep learning techniques and contributes to the growing field of scientific machine learning.

References

Daniel Bashir, George D. Montanez, Sonia Sehra, Pedro Sandoval Segura, and Julius Lauw. An information-theoretic perspective on overfitting and underfitting, 2020.

Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias–variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32): 15849–15854, 2019. doi: 10.1073/pnas.1903070116. URL https://www.pnas.org/doi/abs/10.1073/pnas.1903070116.

Akash Kumar Bhoi, Pradeep Kumar Mallick, Chuan-Ming Liu, and Valentina E. Balas (eds.). *Bio-inspired Neurocomputing*. Springer Singapore, 2021. doi: 10.1007/978-981-15-5495-7. URL https://doi.org/10.1007%2F978-981-15-5495-7.

David Broomhead and David Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. *ROYAL SIGNALS AND RADAR ESTABLISHMENT MALVERN (UNITED KINGDOM)*, RSRE-MEMO-4148, 03 1988.

Feiyu Chen, David Sondak, Pavlos Protopapas, Marios Mattheakis, Shuheng Liu, Devansh Agarwal, and Marco Di Giovanni. Neurodiffeq: A python package for solving differential equations with neural networks. *Journal of Open Source Software*, 5(46):1931, 2020. doi: 10.21105/joss.01931. URL https://doi.org/10.21105/joss.01931.

Salvatore Cuomo, Vincenzo Schiano di Cola, Fabio Giampaolo, Gianluigi Rozza, Maziar Raissi, and Francesco Piccialli. Scientific machine learning through physics-informed neural networks: Where we are and what's next, 2022.

E. M. de Jager. On the origin of the korteweg-de vries equation. 2006. doi: 10.48550/ARXIV.MATH/0602661. URL https://arxiv.org/abs/math/0602661.

P. G. Drazin and R. S. Johnson. *Solitons: An Introduction*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2 edition, 1989. doi: 10.1017/CBO9781139172059.

John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159, 2011. URL http://jmlr.org/papers/v12/duchi11a.html.

Weinan E and Bing Yu. The deep ritz method: A deep learning-based numerical algorithm for solving variational problems, 2017. URL https://arxiv.org/abs/1710.00211.

Weinan E, Jiequn Han, and Arnulf Jentzen. Algorithms for solving high dimensional PDEs: from nonlinear monte carlo to machine learning. *Nonlinearity*, 35(1):278–310, dec 2021. doi: 10.1088/1361-6544/ac337f. URL https://doi.org/10.1088%2F1361-6544%2Fac337f.

Tamara G. Grossmann, Urszula Julia Komorowska, Jonas Latz, and Carola-Bibiane Schönlieb. Can physics-informed neural networks beat the finite element method?, 2023.

Julia Gurieva, Evgenii Vasiliev, and Lev Smirnov. Application of conservation laws to the learning of physics-informed neural networks. *Procedia Computer Science*, 212:464–473, 2022. ISSN 1877-0509. doi: https://doi.org/10.1016/j.procs.2022.11.030. URL https://www.sciencedirect.com/science/article/pii/S1877050922017215. 11th International Young Scientist Conference on Computational Science.

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 Maciej Paszynski, Dieter Kranzlmüller, Valeria V. Krzhizhanovskaya, Jack J. Dongarra, and Peter M.A. Sloot (eds.), *Computational Science – ICCS 2021*, pp. 447–461, Cham, 2021. Springer International Publishing.

Zheyuan Hu, Ameya D. Jagtap, George Em Karniadakis, and Kenji Kawaguchi. When do extended physics-informed neural networks (XPINNs) improve generalization? *SIAM Journal on Scientific Computing*, 44 (5):A3158–A3182, sep 2022. doi: 10.1137/21m1447039. URL https://doi.org/10.1137%2F21m1447039.

Ameya D. Jagtap, Ehsan Kharazmi, and George Em Karniadakis. Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems. *Computer Methods in Applied Mechanics and Engineering*, 365:113028, 2020. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2020.113028. URL https://www.sciencedirect.com/science/article/pii/S0045782520302127.

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

Alexander Koryagin, Roman Khudorozkov, and Sergey Tsimfer. Pydens: a python framework for solving differential equations with neural networks, 2019.

Nikola Kovachki, Zongyi Li, Burigede Liu, Kamyar Azizzadenesheli, Kaushik Bhattacharya, Andrew Stuart, and Anima Anandkumar. Neural operator: Learning maps between function spaces, 2022.

Aditi S. Krishnapriyan, Amir Gholami, Shandian Zhe, Robert M. Kirby, and Michael W. Mahoney. Characterizing possible failure modes in physics-informed neural networks, 2021.

Zaharaddeen Lawal, Hayati Yassin, Daphne Teck Ching Lai, and Azam Idris. Physics-informed neural network (pinn) evolution and beyond: A systematic literature review and bibliometric analysis. *Big Data and Cognitive Computing*, 11 2022. doi: 10.3390/bdcc6040140.

Yuhao Liu, Xiaojian Li, and Zhengxian Liu. An improved physics-informed neural network based on a new adaptive gradient descent algorithm for solving partial differential equations of nonlinear systems, 2022. URL https://doi.org/10.21203/rs.3.rs-2192513/v1.

Lu Lu, Xuhui Meng, Zhiping Mao, and George Em Karniadakis. Deepxde: A deep learning library for solving differential equations. *SIAM Review*, 63(1):208–228, 2021. doi: 10.1137/19M1274067. URL https://doi.org/10.1137/19M1274067.

Siddhartha Mishra and Roberto Molinaro. Estimates on the generalization error of physics-informed neural networks for approximating PDEs. *IMA Journal of Numerical Analysis*, 43(1):1–43, 01 2022. ISSN 0272-4979. doi: 10.1093/imanum/drab093. URL https://doi.org/10.1093/imanum/drab093.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt, 2019.

Preetum Nakkiran, Prayaag Venkat, Sham Kakade, and Tengyu Ma. Optimal regularization can mitigate double descent, 2021.

Guofei Pang, Wen Chen, and Zhuojia Fu. Space-fractional advection–dispersion equations by the kansa method. *Journal of Computational Physics*, 293:280–296, 2015. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2014.07.020. URL https://www.sciencedirect.com/science/article/pii/S0021999114005130. Fractional PDEs.

Swathi Pothuganti. Review on over-fitting and under-fitting problems in machine learning and solutions. *International Journal of Advanced Research in Electrical Electronics and Instrumentation Engineering*, 7: 3692–3695, 09 2018. doi: 10.15662/IJAREEIE.2018.0709015.

M. Raissi, P. Perdikaris, and G.E. Karniadakis. Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations. *Journal of Computational Physics*, 378:686–707, 2019. ISSN 0021-9991. doi: https://doi.org/10.1016/j.jcp.2018.10.045. URL https://www.sciencedirect.com/science/article/pii/S0021999118307125.

Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics informed deep learning (part i): Data-driven solutions of nonlinear partial differential equations, 2017. URL https://arxiv.org/abs/1711.10561.

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

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958. doi: 10.1037/h0042519. URL http://dx.doi.org/10.1037/h0042519.

Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017.

Justin Sirignano and Konstantinos Spiliopoulos. DGM: A deep learning algorithm for solving partial differential equations. *Journal of Computational Physics*, 375:1339–1364, dec 2018. doi: 10.1016/j.jcp.2018.08.029. URL https://doi.org/10.1016%2Fj.jcp.2018.08.029.

N. Sukumar and Ankit Srivastava. Exact imposition of boundary conditions with distance functions in physics-informed deep neural networks. *Computer Methods in Applied Mechanics and Engineering*, 389: 114333, 2022. ISSN 0045-7825. doi: https://doi.org/10.1016/j.cma.2021.114333. URL https://www.sciencedirect.com/science/article/pii/S0045782521006186.

Thiab R Taha and Mark I Ablowitz. Analytical and numerical aspects of certain nonlinear evolution equations. iii. numerical, korteweg-de vries equation. *Journal of Computational Physics*, 55(2): 231–253, 1984. ISSN 0021-9991. doi: https://doi.org/10.1016/0021-9991(84)90004-4. URL https://www.sciencedirect.com/science/article/pii/0021999184900044.

Jian Wang, Xue Pang, Faliang Yin, and Jun Yao. A deep neural network method for solving partial differential equations with complex boundary in groundwater seepage. *Journal of Petroleum Science and Engineering*, 209:109880, 2022. ISSN 0920-4105. doi: https://doi.org/10.1016/j.petrol.2021.109880. URL https://www.sciencedirect.com/science/article/pii/S0920410521014972.

Sifan Wang, Yujun Teng, and Paris Perdikaris. Understanding and mitigating gradient flow pathologies in physics-informed neural networks. *SIAM Journal on Scientific Computing*, 43(5):A3055–A3081, 2021. doi: 10.1137/20M1318043. URL https://doi.org/10.1137/20M1318043.

N. J. Zabusky and M. D. Kruskal. Interaction of "solitons" in a collisionless plasma and the recurrence of initial states. *Phys. Rev. Lett.*, 15:240–243, Aug 1965. doi: 10.1103/PhysRevLett.15.240. URL https://link.aps.org/doi/10.1103/PhysRevLett.15.240.

A Procedure to calculate the values of parameters of 2-soliton solution of KdV

This part shows how we get the answers that $k_1 = -2, k_2 = 4, \eta_1^0 = \ln\frac{1}{3}, \eta_2^0 = \ln\frac{1}{3}$

Proof.

$$\begin{aligned} u(x,0) &= 2(\log f)_{xx}\,|_{t=0} = 6\,\mathrm{sech}^2(x) \\ &\Downarrow \\ \log f\,|_{t=0} &= 3\log(\cosh(x)) + c_1 x + c_2 \\ &= \log\left(\cosh^3(x) \times e^{c_1 x} \times e^{c_2}\right) \\ &\Downarrow \\ f &= 1 + e^{\eta_1} + e^{\eta_2} + e^{\eta_1+\eta_2+A_{12}} \\ &= \frac{1}{8} \times e^{c_1 x} \times e^{c_2} \times \left(e^{-3x} + 3e^{-x} + 3e^{x} + e^{3x}\right) \end{aligned} \tag{13}$$

As $\eta_i = k_i x + \eta_i^0$, we firstly have to find a group of k_1, and k_2 that satisfy $k_1, k_2, k_1 + k_2$ can be $-3 + c_1, -1 + c_1, 1 + c_1, 3 + c_1$ in some order. This is not difficult and we can find such a group in which $k_1 = -2, k_2 = 4$ and then $c_1 = 1$.

According to $e^{A_{ij}} = (\frac{k_i - k_j}{k_i + k_j})^2$, it is easy to find that $e^{A_{12}} = 9$.

Subsequently, we shall determine the values of η_1 and η_2, contingent upon the confirmation of c_2's value. The following steps outline how to obtain c_2 through the utilization of the constant term detailed above.

$$\begin{aligned} \frac{1}{8} \times e^{c_2} \times 3 &= 1 \\ \frac{1}{8} \times e^{c_2} &= e^{\eta_1^0} \\ \frac{1}{8} \times e^{c_2} \times 3 &= e^{\eta_2^0} \end{aligned} \tag{14}$$

Now it is easy to derive that $e^{c_2} = \frac{3}{8}$ and then we find:

$$\eta_1^0 = \ln\frac{1}{3}$$

$$\eta_2^0 = \ln\frac{1}{3}$$

Overall, we calculated that $k_1 = -2, k_2 = 4, \eta_1^0 = \ln\frac{1}{3}, \eta_2^0 = \ln\frac{1}{3}$, which makes the 2 solitons interact when $t = 0$

□

B Procedure to calculate the values of parameters of 3-soliton solution of KdV

This part shows how we get the answers that $k_1 = 2, k_2 = 4, k_3 = -6, \eta_1^0 = \ln\frac{3}{2}, \eta_2^0 = \ln\frac{3}{5}, \eta_3^0 = \ln\frac{1}{10}$. The work in this piece is harder than the process of the 2-soliton case, but we can still derive the values.

Proof.

$$\begin{aligned}
u(x,0) &= 2(\log f)_{xx}\,|_{t=0} = 12\,\mathrm{sech}^2(x) \\
&\Downarrow \\
\log f\,|_{t=0} &= 6\log(\cosh(x)) + c_1 x + c_2 \\
&= \log\left(\cosh^6(x) \times e^{c_1 x} \times e^{c_2}\right) \\
&\Downarrow \\
f &= 1 + e^{\eta_1} + e^{\eta_2} + e^{\eta_3} + e^{\eta_1+\eta_2+A_{12}} + e^{\eta_2+\eta_3+A_{23}} + e^{\eta_3+\eta_1+A_{31}} + e^{\eta_1+\eta_2+\eta_3+A_{12}+A_{23}+A_{31}} \\
&= \frac{1}{64} \times e^{c_1 x} \times e^{c_2} \times (e^{-6x} + 6e^{-4x} + 15e^{-2x} + 15e^{2x} + 6e^{4x} + e^{6x} + 20)
\end{aligned} \tag{15}$$

As $\eta_i = k_i x + \eta_i^0$, we firstly have to find a group of k_1, k_2 and k_3 that satisfy $k_1, k_2, k_3, k_1+k_2, k_2+k_3, k_3+k_1, k_1+k_2+k_3$ can be $-6+c_1, -4+c_1, -2+c_1, 0+c_1, 2+c_1, 4+c_1, 6+c_1$ in some order. This is not difficult and we can find such a group in which $k_1 = 2, k_2 = 4, k_3 = -6$ and then $c_1 = 0$.

According to $e^{A_{ij}} = (\frac{k_i - k_j}{k_i + k_j})^2$, it is easy to find that $e^{A_{12}} = \frac{1}{9}, e^{A_{23}} = 25, e^{A_{31}} = 4$.

Next we shall find the values of $\eta_1, \eta_2,$ and η_3. However, we still need to confirm the value of c_2 before we continue the calculation. Below are the steps to find c_2 by using the constant term above:

$$\begin{aligned}
\frac{1}{64} \times e^{c_2} \times 15 &= e^{\eta_1^0} \\
\frac{1}{64} \times e^{c_2} \times 6 &= e^{\eta_2^0} \\
\frac{1}{64} \times e^{c_2} \times 1 &= e^{\eta_3^0} \\
\frac{1}{64} \times e^{c_2} \times 20 &= 1 + \frac{1}{9} \times 25 \times 4 \times e^{\eta_1^0+\eta_2^0+\eta_3^0} \\
&= 1 + \frac{100}{9} \times \frac{90}{64^3} \times (e^{c_2})^3
\end{aligned} \tag{16}$$

By letting $k = \frac{1}{64} \times e^{c_2}$, we have the form that:

$$20k = 1 + 1000k^3,$$

and one solution of k is $\frac{1}{10}$.

Therefore, we can calculate that $\eta_1^0 = \ln\frac{3}{2}, \eta_2^0 = \ln\frac{3}{5}, \eta_3^0 = \ln\frac{1}{10}$.

□

C Pseudocode for ADAM

Algorithm 1 ADAM

Input : A constant vector $\mathbb{R}^d \ni \xi \mathbf{1}_d > 0$, parameters $\beta_1, \beta_2 \in [0,1)$, a sequence of step sizes $\{s_t\}_{t=1,2..}$, initial starting point $\boldsymbol{x}_1 \in \mathbb{R}^d$, and we are given oracle access to the gradients of $f : \mathbb{R}^d \to \mathbb{R}$.
https://www.overleaf.com/project/627cd8b4638ce3cd4634561d
▷ f is typically the loss function that we are trying to minimize.
▷ Hence, typically $\boldsymbol{x}$ are the weights we are optimizing over
▷ ,and then d would be the total number of trainable weights in the model.

function ADAM($\boldsymbol{x}_1, \beta_1, \beta_2, s, \xi$)
  Initialize : $\boldsymbol{m}_0 = \mathbf{0}$, $\boldsymbol{v}_0 = \mathbf{0}$
  for $t = 1, 2, \ldots$ **do**
    Oracle sends $\mathbf{g}_t$ s.t $\mathbb{E}[\mathbf{g}_t \mid \{\mathbf{x}_i\}_{i=1,\ldots,t}] = \nabla f(\mathbf{x}_t)$
    $\mathbf{m}_t = \beta_1 \mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t$
    $\mathbf{v}_t = \beta_2 \mathbf{v}_{t-1} + (1-\beta_2)(\mathbf{g}_t \circ \mathbf{g}_t)$
    $\mathbf{V}_t = \mathrm{diag}(\mathbf{v}_t)$
    $\mathbf{x}_{t+1} = \mathbf{x}_t - s_t \left(\mathbf{V}_t^{\frac{1}{2}} + \mathrm{diag}(\xi \mathbf{1}_d)\right)^{-1} \mathbf{m}_t$
  end for
end function

D Pseudocode for Physics Informed Neural Network

Algorithm 2 Physics Informed Neural Network

Input : A nonlinear PDE $\partial_t u(x,t) + \mathcal{L}u(x,t) = 0$, the initial condition $u(x,0) = u_0(x), x \in \Omega$, the boundary condition $u(x,t) = f(x,t), (x,t) \in \partial\Omega \times [0,\mathrm{T}]$. Also the space interval Ω and the time interval $[0,\mathrm{T}]$.

w: Neural Network Layer Width
d: Neural network depth
size: Data size
epoch: Training epoch
lr: learning rate of the optimizer

function PINN(PDE, $w, d, size, epoch, lr$)
  Initialization : $net \xleftarrow{w,d}$ *initialize neural network*; $Optimizer \xleftarrow{lr}$ *Adam*;
  Generator : $X \xleftarrow{\Omega,\mathbf{T}}$ *sampling x and t*;
  Model : based on net and boundary/initial condition
  for i in epoch **do**
    clean gradient of optimizer
    $\mathbf{x}$ = generator($size$)
    loss = Loss(x)
    Update parameters of model
  end for
end function