Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Masterarbeit

Learning Free-Surface Flow with Physics-Informed Neural Networks

Marcel Hurler

Course of Study:	Computer Science
Examiner:	Prof. Dr. rer. nat. Dirk Pflüger
Supervisor:	Raphael Leiteritz, M.Sc.
Commenced:	July 24, 2020
Completed:	January 24, 2021

Abstract

This thesis examines the application of physics-informed neural network to solve free-surface flow problems modeled with the shallow water equations. Physics-informed neural network allow training of a surrogate model that resembles the latent solution of an underlying partial differential equation, without using any training data sampled from experiments or numerical simulations. The shallow water equations are an approximation of the Navier stokes equations and serve as a model to many environmental flow problems including dam-breaks, floods, and tsunami propagation. The equations form a non-linear system of hyperbolic partial differential equations that describe the evolution of a fluid's depth and momentum through time. Contrary to other models for free-surface flow, where the exact location of the free surface is only given implicitly as an isosurface and needs reconstruction, here, the depth directly yields its location. One characteristic of the shallow water equations is the formation of steep wavefronts and discontinuities. The thesis examines four state-of-the-art techniques to improve accuracy and training speed and discusses their behavior on three initial value problems. These include the famous idealized dam-break and two depth perturbations, one above a flat and one above varying bathymetry. For each of the scenarios, an inspection of suitable network architectures was considered. Additionally, three different formulations of the physics-informed neural network are presented and tested, where one approach implicitly fulfills the mass conservation and thus eliminates one equation of the system. The results show, that it is possible to train a surrogate model with a relative L^2 error of less than 10^{-4} compared to a solution computed by a high-resolution numerical solver in case of a moderate steepening of wavefronts. A relative error close to 10^{-3} can be achieved for the dam break problem, where the initial conditions are discontinuous, and the solution contains shocks that propagate over time. Additionally, it shows that training with bathymetry is possible and the learned depth approximates the varying underground without any noticeable difference.

Contents

List of Figures

List of Tables

Acronyms

ADAM adaptive moment estimation method. 27, 28, 36, 40, 60

ANN artificial neural network. 14, 15, 22, 24, 27, 29

L-LAAF layer-wise locally adaptive activation functions. 38, 47, 48, 51, 52, 53, 54, 55, 56, 60

LHS latin hypercube sampling. 32, 33, 45

LRA learning rate annealing. 36, 38, 44, 45, 47, 48, 51, 52, 55, 56, 60

MSE mean squared error. 26, 39

N-LAAF neuron-wise locally adaptive activation functions. 38, 47, 48, 49, 51, 52, 53, 55, 56, 57, 60

PDE partial differential equation. 14, 15, 19, 24, 25, 29, 32, 34, 35, 36, 38, 44, 50, 53, 60, 61

PINN physics-informed neural network. 3, 7, 9, 15, 24, 25, 29, 32, 33, 34, 35, 36, 37, 42, 43, 44, 45, 46, 47, 50, 53, 55, 60, 61

SIREN sinusoidal representation network. 37, 42, 43, 47, 48, 52, 53, 56

SWE shallow water equations. 7, 14, 15, 18, 19, 20, 21, 22, 24, 32, 33, 34, 35, 42, 60, 61

1 Introduction

1.1 Motivation

Artificial neural networks (ANNs) have reached many scientific disciplines ranging from classification and generative tasks in well-known fields like computer vision and natural language processing to recent applications in material science [SMBM19] and protein folding [JEP+20]. One reason for the splendid success of ANNs stems from their ability to apporixmate a large space of nonlinear functions. This is essential to solving many real-world problems since they often require building a complex model to describe the relationship between variables of interest accurately. An example is the classification of human faces based on an input image which results in a highly nonlinear decision boundary in the space of all RGB-images. Some network architectures, e.g. the feed-forward architecture fulfill the universal function approximation theorem. In short, this means that there exists an approximation for any continuous function over the Euclidean space. Recently, Raissi et al. [RPK19] exploited this property and invented a framework for solving forward and inverse problems involving *partial differential equations (PDEs)* using ANNs. In this case, the forward problem amounts to solving a PDE with given initial and boundary conditions. The inverse problem involves inferring parameters of a PDE by observing measurements in the spatio-temporal domain. Since PDEs are used to describe many processes in nature, the contribution has applications in many scientific fields, e.g. natural sciences and engineering [RYK18; WWW20; WZ20]. This work concentrates on the forward problem and investigates the framework for solving free-surface flow problems described by the 1D *shallow water equations (SWE)*.

Free-surface flow is a branch in fluid mechanics that focuses on the interface between two or more fluids that strongly differ in density, e.g. between water and air. The field of fluid mechanics is dominated by the Navier Stokes equations which is a system of PDEs that describe the behavior of fluids as an interplay of quantities like velocity, pressure, and temperature. The equations are known to be hard to solve, and over the years, the equations were simplified and specialized for, typically idealized, use cases. One such simplification is the shallow water equations, which is a nonlinear, hyperoblic type of PDEs that describes the movement of one or more free surfaces in scenarios where the horizontal scale is much larger than the vertical scale. Typical use cases range from tsunami forecasting to simulations of dam breaks and floods. They are typically solved with classical numerical methods, as analytical solutions only exist for limited cases.

Numerical methods typically fall under one of three types: finite elements, finite volumes, and finite differences, that all involve a discretization of the considered domain. That means they only compute the solution at discrete point locations, e.g. wind speed, temperature, and pressure every ten kilometers for a global weather forecast. For gathering values in between, interpolation is used, which rarely obeys the rules of the underlying PDE. A suitable discretization is crucial to get representative results. Typically, the discretization process is highly problem dependent and often involves manual inspection and adaption. Instead, the discretization process turns into a less restrictive sampling strategy when training an ANN to solve the PDE. The result is an analytical approximation to the PDE's latent solution which naturally provides a result at any point in the spatio-temporal domain without the need to interpolate. Contrary to classical machine learning tasks, no careful preprocessing of labeled training data is necessary, as the information mainly comes from the PDE itself and is enforced undisturbedly via a unique loss term. Here, the training data typically contains few labeled data of sampled initial and boundary conditions and many points in the spatio-temporal domain to enforce the rules of the PDE. Apart from its benefits, the main practical drawback is that learning the PDE is currently more time consuming compared

to solving it with well established and highly optimized numerical methods. However, recently many techniques were developed to speed up the convergence process [JKK19; JKK20a; MB20; SMB+20; WTP20; WYP20; WZ20]. Where some are investigated throghout this thesis to improve the achieved results.

To my best knowledge, solutions to the SWE within the PINN framework have not yet been presented. The only work similar to mine was recently published by Wessels et al. [WWW20] which considers the solution of the implicit Euler equations in an updated Lagrangian formulation. They use the PINN as a spatial ansatz function before Runge Kutta stages, which was previously described by Raissi et al. [RPK19], and therefore solve the equations in a stepwise approach. As their formulation considers individual fluid particles, the free surface is defined implicitly, and a reconstruction of its exact location is possible. A method that combines the SWE with neuronal networks to create a surrogate model has been proposed recently by Zhang et al. [ZZX+20]. They also use a stepwise approach to solve the 2D SWE with the help of a convolutional neural network. However, their training is solely based on numerical simulation results and does not include the laws of the underlying PDE. Also, their architecture only computes the solution at discrete points.

1.2 Outline of the Thesis

The thesis is divided into six chapters, where Chapter 2 introduces the shallow water equations in three sections. The first section gives a brief overview of modeling assumptions and resulting fields of applications. The following section presents the equations and describes general characteristics. The third section concerns classical methods to compute solutions and gives a brief overview of the methods used to evaluate the results achieved in this work.

Chapter 3 presents the framework of PINNs and lies the foundation for the methods that were used to solve the SWE. The chapter is divided into three sections, where the first section provides an introduction to ANNs with a focus on feed-forward networks. The second Section considers the training procedure of typical regression problems and describes the base gradient descent method along with a broadly used optimization. The last section builds on the previous two and defines the PINN framework and its training procedure similar to how it was originally described by Raissi et al. [RPK19].

Chapter 4 investigates the application of the PINN to the SWE. General network and training assumptions are defined in the first section. The following section presents four optimization methods that were examined in this work to improve the accuracy and training speed of the learned models.

Chapter 5 presents the results achieved with the methods described in the previous chapter after solving three initial value problems with different levels of complexity. The problems involve an initial depth perturbation over a flat bathymetry, a dam break scenario, and a depth perturbation with varying bathymetry.

Chapter 6 concludes the work and shows prospects for future work.

2 Shallow Water Equations

This chapter presents the SWE along with their properties and classical solution methods. The first section motivates the equations and provides examples for areas of application. The following section introduces the equations for the 1D case in the conservative and non-conservative form and explains the terms and modeling assumptions. The last section concentrates on solving the equations and provides an overview of numerical and analytical methods.

2.1 Basic Principles and Applications

Shallow-water flow is part of the broad field of fluid mechanics, which includes the theory of fluid statics and fluid dynamics. The dominating equations in this field are the three-dimensional Navier Stokes equations. They are derived from the conservations of mass and momentum and can describe all types of fluids in terms of fundamental properties, e.g. velocity, pressure, and temperature. While the classical Navier Stokes equations consider the flow inside a single fluid medium, the branch of free-surface flows focuses on the interface between fluids. If the difference in density between two fluids is high enough, an interface emerges and forms the free surface. Typical examples are between a gas and a fluid such as water and air, but also other combinations are possible and are conceptually similar. Free surfaces appear on different scales. On a smaller scale, air entrainments, i.e. the creation of bubbles and foaming can be observed. On a larger scale, the free surface forms the surface of rivers, lakes, and oceans. However, solving these equations with the same fidelity on arbitrary scales is infeasible due to the computational complexity. In many cases the benefit of the high precision does not justify the computational effort. Therefore the equations have been simplified, and tailored to idealized cases.

The shallow water equations (SWE) are one of such simplifications [Cli] and consider flows where the horizontal scale of the fluid medium is much larger than the vertical scale, i.e. the depth of the fluid beneath the free surface. This restriction allows two simplifications to the original three-dimensional model [Kat19]: (1) Observations show, that particle orbits under a graviation wave transform from circles to horizontally stretched ellipses. Details in the vertical velocity can therefore be neglected and integrated over the depth. In the SWE they become the vertical movement of the free surface. (2) Due to the small depth, it can be shown, that the pressure gradient is almost hydrostatic, i.e. increases linearly with depth. That suggests that the only driving force is the gravitation, and the horizontal velocity is assumed the same for a single column of fluid. The free surface is approximated by a single, infinitely thin surface, that is described by a scalar-valued heightfield. A downside of this representation is that it does not allow two values at the same location, and therefore air entrainment and overhangs i.e. braking waves can not be modeled. Contrary to what the term shallow suggests, such flows do not only appear on a smaller scale like streams, ponds, and puddles. Also, larger, up to global scale phenomenons can be modeled successfully with the SWE.

Typical fields of applications are:

- Tsunami Propagation — Tsunamis are ocean waves with wavelengths of several hundred kilometers that can create severe damage to infrastructure and population around coastlines [LPF+13]. The main driving sources are earthquakes on the seafloor, which lead to disturbance of the water surface that propagates due to the gravitational force. Models have been proposed based on the SWE [MS10] and are used in simulation software today [Cla20; RD08].

- Storm Surges — Extreme weather phenomenons such as hurricanes, cyclones, or typhoons produce strong winds, which cause long and large waves traveling to coastlines. The consequences are flooding of urban areas and erosions of shores. Due to climate change, these hazards tend to increase. The SWE allow modeling and forecasting [WLB+92] of such events, which can help to introduce suitable counter measurements.
- Dam Breaks — Dams are used as barriers to prevent floods and serve as water reservoirs for human consumption, and electrical energy, i.e. hydropower. Dam failure can cause a major disaster, ranging from damaged buildings to endangerment of human lives [Ger05]. While there are analytical solutions for idealized cases, numerical methods based on the SWE offer a more general application spectrum [Pen12].

2.2 The 1D Shallow Water Equations

Many realistic problems require to solve the 2D shallow water equations and cannot be simplified to a single spatial dimension. However, the experience shows that due to high frequencies, which emerge when a wave forms a steep front and tends to brake or during shock propagations in case of a dam break, even the 1D equations are not trivial to solve. Therefore in this thesis 2D problems are not considered.

The 1D shallow water equations are a hyperbolic non-linear coupled system of PDE. Their conservative form is as follows:

$$\frac{\partial h}{\partial t} = -\frac{\partial hu}{\partial x} \tag{2.1}$$

$$\frac{\partial hu}{\partial t} = -\frac{\partial hu^2 + gh^2/2}{\partial x} - gh\frac{\partial b}{\partial x} \tag{2.2}$$

Expanding all spatial and temporal derivatives and using the first equation to replace $\frac{\partial h}{\partial t}$ in the second equation, yields the following non-conservative form:

$$\frac{\partial h}{\partial t} = -h\frac{\partial u}{\partial x} - u\frac{\partial h}{\partial x} \tag{2.3}$$

$$\frac{\partial u}{\partial t} = -u\frac{\partial u}{\partial x} - g\left(\frac{\partial h}{\partial x} + \frac{\partial b}{\partial x}\right) \tag{2.4}$$

The depth of the fluid is denoted by $h = h(x,t)$ and measures the distance from the impermeable *bathymetry* (seafloor) $b = b(x)$ to the free surface at a specific location $x \in \Omega \subset \mathbb{R}$ at time $t \in [0,T]$. It is worth mentioning that the term $\frac{\partial b}{\partial x}$ vanishes and simplifies the equations, when the bathymetry is flat. The horizontal velocity at the same location is given by $u = u(x,t)$. In this representation, the only driving force is the gravitation, here denoted by g. Modeling other physical quantities such as viscosity, friction, or Coriolis force is also possible but not considered in this case. For

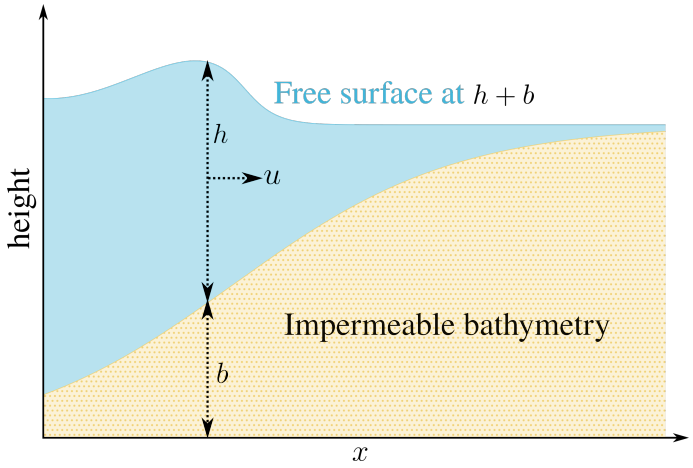

Figure 2.1: Illustration of the variables described in the shallow water equations.

better understanding, Figure 2.1 illustrates an example. Here, equations 2.1 and 2.3 derive from the conservation of mass and equations 2.2 and 2.4 from the conservations of momentum respectively. The derivation from the conservation laws is not part of this work. For a detailed investigation, the reader is referred to the great thesis of Jhon Jakeman [Jak06]; or for a broader view, Nikolaos Katopodes derives the equations in chapter 4 of his book [Kat19], including additional modeled properties.

A property of the SWE is the steepening of the wavefront that emerges when the wave propagates. In reality, this effect can typically be observed when the wave approaches the shore. Because of the increasing bathymetry, water particles at the wavefront near the bottom slow down while particles closer to the top of the wave keep their horizontal speed. This allows the crest of the wave to catch up with the wavefront and form the steep face of the wave. The deceleration of the lower particles also induces a vertical motion which increases the height of the wave and decreases its wavelength. In reality, this effect leads to the breaking of the wave, which can not be described by the SWE. Although the SWE model the steepening of the wavefront in case of an increasing bathymetry, eventually all waves form a steep front independent of the bottom profile. This behavior can be observed in Figure 2.2 and is a weak point of the model and does not reflect observations in reality.

Considering the conservative form of the equations it shows, that the horizontal velocity is not propagated directly through time. Instead, the time derivative of the *momentum*, sometimes also denoted by *discharge*, expressed by hu, is concerned, which measures the flow rate of fluid passing a point. The reason for this is that velocity is not a conserved quantity regarding the conservation laws. A problem with non-conservative equations shows during discretization when a quantity is discontinuous—e.g. this is the case for the dam break problem as we will see later in Section 5.2. To make this more clear, consider the spatial derivative in equation 2.3 $h\frac{\partial u}{\partial x}$. Approximating this term with finite differences requires to pick at least two u's and one h at locations of the discretization. For example when using forward differences the approximation is $h_j \frac{u_{i+1}-u_i}{\Delta x}$. While the choice of u_{i+1} and u_i is straight forward, the selection of h_j is not and leads to different results, especially when the supporting points of u_i and u_{i+1} are adjacent to the discontinuity.

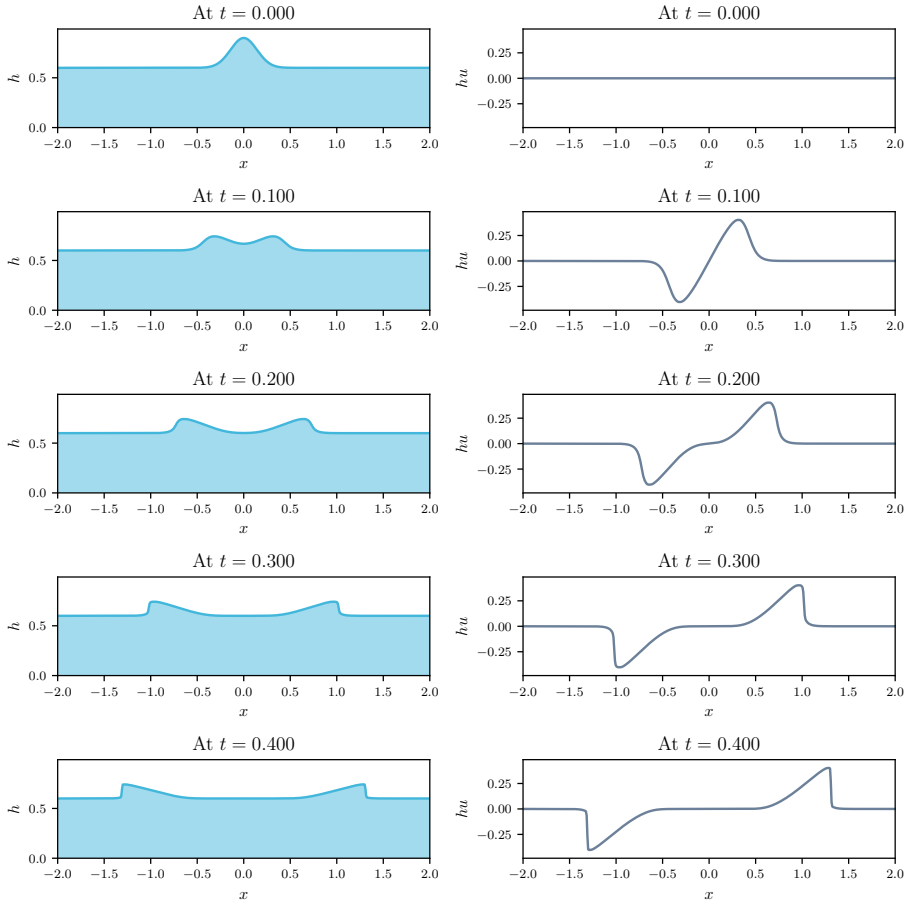

Figure 2.2: Evolution of an initial depth perturbation at the center of the domain over a flat bathymetry. The SWE are solved with the Pyclaw [KMA+12] software, where the gravity g is set to 9.8. The left column shows the water depth h and the right column the momentum hu.

2.3 Solutions to the 1D Shallow Water Equations

The non-linear and hyperbolic characteristics of the SWE only allow analytical solution for idealized cases and thus numerical treatment is required. For an overview of analytical solutions, see [DLK+13]. The broad range of proposed numerical methods covers finite volume methods [ZR03], finite elements methods [BC96], and spectral methods [SK11]. Commonly, these methods are optimized to withstand shocks which can be present at beginning of a simulation, for example, when the initial condition contains a discontinuity (e.g. in case of a dam break). Or later, when the wavefront steepens as described in the previous section. Other enhancements consider the dampening of oscillations which are typical for higher order methods and the avoidance of entropy violation. Jakeman [Jak06] provides an overview of methods in his thesis. In chapter 4, he describes the principles of the finite volumes and finite elements method regarding the SWE and outlines improvements to these which concern the forementioned optimizations.

The numerical solutions concerned in this work serve as validation data during the training of the ANN-based model described in section 3.2.1. Since I chose Python in this work because of its simple access to modern deep learning libraries and great plotting capabilities, it seemed natural to search for a framework that provides solutions to the SWE as well in that language. After inspecting available frameworks, including mattflow [Mat], ANUGA [RC01] and pyswashes [DLK+13], the Clawpack (Conservation Law Package) [Cla20] turned out to be most appropriate because of its good documentation and the possibility to model various 1D and 2D examples; in contrast, the first two only support 2D and the latter is limited to idealized analytical solutions. Clawpack provides a collection of finite volume methods for linear and non-linear hyperbolic systems of conservation laws for the Python and Fortran programming languages. The methods used in this work are part of the Pyclaw software [KMA+12], which is part of Clawpack and provides a more Python related interface (other parts of the software require to implement portions in Fortran) and parallelism that can scale up to thousands of nodes. The examples presented in this work are of two kinds: Those which incorporate a varying bathymetry and those where the bathymetry is assumed to be flat. For each of the two types, Pyclaw provides a different numerical treatment. The following briefly describes their underlying concept.

For solving the SWE, Clawpack relies on Godunov's method for non-linear conservation laws [LeV02]. The originally first-order finite volume scheme can handle shock waves in an "upwind" manner, even for systems of equations where information propagates in both directions. The idea is to approximate each cell of the finite volume discretization by a constant-valued function that returns the cell's average at each timestep. To compute the solution at a following timestep it is necessary to compute the fluxes at each cell interface, i.e. the right-hand sides of Equations 2.1 and 2.2. The Godunov method formulates each cell interface as a Riemann problem (a generalization of the dam break problem, see section 5.2) that can be solved analytically, even if the equations are non-linear (via the method of characteristics). The result is averaged for each cell and forms the solution of the next timestep. As the exact solution to the Riemann problem typically is very time-consuming, approximations were invented to speed up the solution process. One such approximation is the HLLE method [TT09], which I applied for cases without bathymetry in this work, as suggested in the Clawpack documentation [Cla]. Similarly, I chose a second-order-accurate solver that uses the f-wave approach [LG08] for cases with bathymetry. The technical details would go beyond this thesis, and the reader is referred to the given literature. In all presented scenarios, I chose a spatial discretization with 500 cells. The number of waves is set to 2 in cases of the f-wave approach.

3 Pysics Informed Neural Networks

This chapter introduces physics-informed neural networks (PINNs) and sets the foundations of how they can be used to solve the SWE. Before going into details, a brief introduction to artificial neural networks (ANNs) in the first section will provide the basis of the following methods and explain general terminologies that appear throughout the work. The subsequent section describes how PINNs can solve PDEs of general form.

3.1 Artificial Neural Networks

In today's machine learning, ANNs have established themselves, besides other algorithms, as a useful and versatile tool. While their invention reaches back to the year 1958 [Ros58], it was a long way until they could show off their full potential. Many theoretical developments including the invention of the *backpropagation* algorithm [Wer75], described in Section 3.2.2, and the discovery of problem-specific architectures [HS97; KSH17] were made to improve their training and generalization performance. However, a dominant factor for their breakthrough results increasing processing power, especially development of fast GPUs, and the availability of large datasets necessary to cover the example space of many relevant problems during training, e.g. classification of images or natural speech and language. Also, the easy availability of open-source software frameworks, e.g. PyTorch [PGM+19] and TensorFlow [ABC+16] which allow fast and efficient development of new models, provide a straightforward way for applications in industry and research.

As the name suggests, the inspiration for ANNs originally came from the biological brain, however the functionality is different in many ways. Like its natural counterpart, it is a collection of connected units, called *artificial neurons*; see figure 3.1. However, where a natural neuron receives signals from other neurons through its dendrites and sends signals to other neurons connected to its axon based on a complex chemical process, artificial neurons only vaguely imitate that behavior. An artificial neuron computes a weighted sum of its inputs $x_1, \dots, x_n$ and weights $w_1, \dots, w_n$ and an offset value b called bias. Afterwards, it applies a non-linear function σ, called *activation function*, to this sum and returns its outcome, here denoted by a:

$$a := \sigma\left(\sum_{i=1}^{n} w_i x_i + b\right) = \sigma(w^\top x + b) \tag{3.1}$$

The activation function is there for two reasons: first, it normalizes the output and prevents a continuous increase or decrease when the outcomes propagate from one neuron to another inside the network. Second, its non-linearity is the key to the approximation quality. Omitting the activation function reduces the network to a linear function over the space of its inputs, which discards the property of a general function approximator and drastically limits its applications. Typical activation functions are the hyperbolic tangent function tanh or the sigmoid function, a generalization of the logistic function.

Over time, many network architectures emerged for various use cases. The most popular and only one concerned in this work is the fully connected *feed-forward network* that was first introduced by Rosenblatt [Ros58]. Figure 3.2 provides a visual representation. It follows a layer-wise approach, where neurons arrange in multiple layers. The number of layers minus one describes the depth.

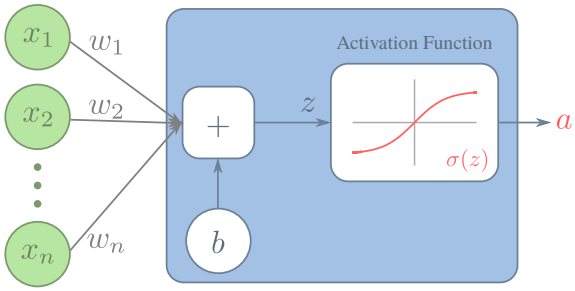

Figure 3.1: A single artificial neuron

Because of the full connection property, each neuron in layer l connects with all neurons in the subsequent layer $l+1$. The layers classify into three classes: *the input layer*, *hidden layer*, and *output layer*. We can think of the input layer as constant output neurons that solely serve as a placeholder to feed the inputs of the network to the neurons in the first hidden layer. The term hidden describes that it is commonly not trivial what exactly neurons in this layer learn or represent. However, it is proven that the complexity and approximation capability increases with the number of layers in this class [KB20; KL19]. The number of neurons in the hidden layer denotes the width of the network. The output layer yields the outcome of the network and depending on the desired output range, its neurons usually omit the activation function. So, for an input $x \in \mathbb{R}^n$ and output $y \in \mathbb{R}^m$ the feed-forward network f_θ of depth k and width p describes the following function:

$$f_\theta(x) = y = \sigma(W^k \underbrace{\sigma(\dots\sigma(W^2\sigma(W^1x + b^1) + b^2) + \dots)}_{a^{(k-1)}} + b^k) \tag{3.2}$$

Here $W^1 \in \mathbb{R}^{p\times n}$ and $b^1 \in \mathbb{R}^p$ denote the matrices which contain all weights and the biases of neurons in the first layer. Similarly, $W^l \in \mathbb{R}^{p\times p}$ and $b^l \in \mathbb{R}^p$ denote the weights and biases of neurons in the hidden layers $2 \le l \le k-1$. And $W^k \in \mathbb{R}^{m\times p}$ and $b^k \in \mathbb{R}^m$ contain the weights and biases of the neurons in the output layer. In contrast to the definition of the neuron, here, the activation function σ operates element-wise. The output of the layer l is denoted by a^l and we define $a^0 := x$. In terms of regression and classification problems, the outcome y is often called the network's prediction. The parameters of the network $\theta = (W^1, \dots, W^k, b^1, \dots, b^k)$ are all weights and biases of all artificial neurons in all layers. Depending on their adjustments, the network alters the mapping of input to output. The training process, described in the next section, exploits that flexibility and steers it to fulfill the training goal.

3.2 Training

3.2.1 Loss Function and Minimization

This section focuses primarily on solving typical regression problems. We will later see that solving a PDE with PINNs also falls into this category. A regression problem generally concerns finding a function that can accurately describe the relationship between a dependent variable and an

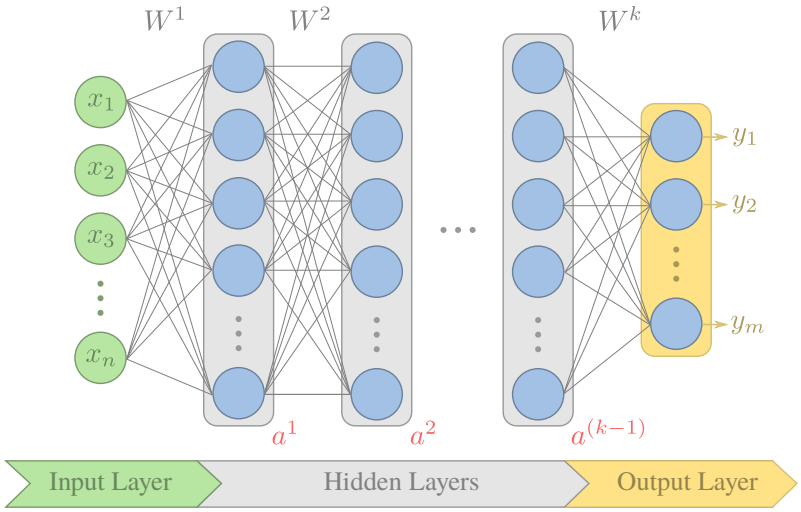

Figure 3.2: Illustration of a feed-forward network

independent one. Consider a labeled dataset $\{(x^i, y^i)\}_{i=1}^D$ where the data points $x^i \in \mathbb{R}^n$ represent the independent variable and corresponding labels $y^i \in \mathbb{R}^m$ denote the dependent variable. Here, we concentrate on the feed-forward network $f_\theta : \mathbb{R}^n \to \mathbb{R}^m$ with its parameters θ as the describing function. A common method of solving such problems is to reframe them as optimization problems. In this setting, a positive and scalar-valued function called *objective function*, often also denoted by *loss function* or *cost function* Ψ is used to penalize deviations of the describing function from the dataset. The best describing function is then minimizing Ψ with respect to the network parameters. The optimal parameter θ^* can thus be expressed by:

$$\theta^* = \arg\min_\theta \Psi(\theta) = \arg\min_\theta \frac{1}{D} \sum_{i=1}^{D} \psi(f(x^i;\theta), y^i) \tag{3.3}$$

Here ψ is also a nonnegative scalar function that measures the error between the network's outcome $f(x^i;\theta)$ to a given data point x^i and its true label y^i. In contrast to equation 3.2, where the parameters are fixed, here the network $f(x;\theta)$ additionally depends on θ. Evaluating the network for a set of parameters and computing the loss function is called computing the *forward pass*. The main loss function that is used in this work is the *mean squared error (MSE)* loss, where ψ measures the error in the squared L^2 norm $||\cdot||_2^2$.

The minimization of the loss function, is called *training* or often vaguely described as *learning*. Typically, the dataset used to train the network, denoted by *training data* is just a small sample from the whole range of possible inputs. To measure the generalization performance of the learned model, typically a different, non overlapping dataset called *validation data* is sampled from the inputs space. The error measured by the loss function on the training data is called *training error* and similarly the error on the validation data is called *validation error*. The minimization can be achieved by many optimization algorithms. Many of these algorithms rely on a gradient descent based approach. Today many advanced versions of this procedure are used. For an overview of gradient descent

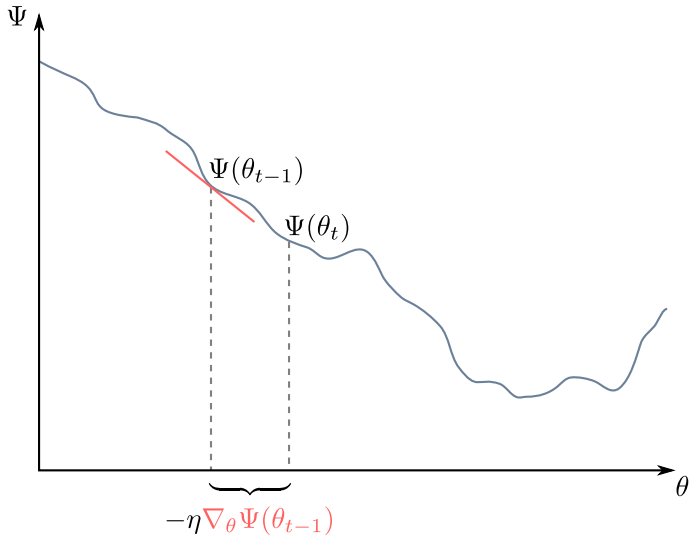

Figure 3.3: Illustration of the gradient descent method for a one dimensional parameter.

based methods, see [Rud16]. The following describes the standard gradient descent method, also known as *batch gradient descent method*. Subsequently, an improved variant, the widely applied *adaptive moment estimation method (ADAM)* [KB17] is presented.

Standard gradient descent is an iterative method. Starting from an initial parameter configuration, each iteration updates the parameters of the network in direction of the steepest descent—which is the negative gradient. The method reads:

$$\begin{aligned} \theta_0 &\leftarrow \text{initial guess} \\ \theta_t &= \theta_{t-1} - \eta \nabla_\theta \Psi(\theta_{t-1}) \end{aligned} \tag{3.4}$$

Figure 3.3 illustrates the procedure for a one-dimensional parameter. Here, η denotes the learning rate, which scales the step size in direction of the steepest descent. A parameter, such as η, that is not optimized during the training process is called *hyperparameter*. The learning rate has a large impact on the convergence of the algorithm. If it is too small, the algorithm converges slowly and may get stuck in a local minimum. However, if it is too large, it may skip the global minimum and potentially even diverge. Especially with ANNs as modeling functions, the loss function is often non-convex with respect to the network parameters and has many local minima.

Through the years, the deep learning community presented improvements to the standard gradient descent method [DHS11; Hin; RHW86]. That also includes the ubiquitous stochastic and memory saving variants, i.e. *stochastic gradient descent* and *mini-batch stochastic gradient descent*. However, the models presented in this work can be trained on a single consumer graphics card, and thus these methods are not further described here.

In the year 2014, Kingma et al. [KB17] presented ADAM, which is a combination of two techniques, the RMSProp algorithm [Hin], a memory-efficient version of the former Rprop algorithm [RB92] and the gradient descent with momentum first presented by Rumelhart et al. [RHW86]. In each step t, ADAM uses an exponentially decaying average of the first moments m_t and second moments v_t of the gradient $g_t := \nabla\Psi(\theta^t)$ to update the parameters. The update rule reads:

$$\begin{aligned} m_t &= \beta_1 \cdot m_{t-1} + (1-\beta_1) \cdot g_t \\ v_t &= \beta_2 \cdot v_{t-1} + (1-\beta_2) \cdot g_t^2 \\ \hat{m}_t &= \frac{m_t}{1-\beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1-\beta_2^t} \\ \theta_t &= \theta_{t-1} - \frac{\eta}{\sqrt{\hat{v}_t}+\epsilon}\hat{m}_t \end{aligned} \tag{3.5}$$

Throughout this work, the hyperparameters are fixed to their suggested values: $\beta_1 = 0.9$, $\beta_2 = 0.999$ and $\epsilon = 10^{-8}$. As in the previous method 3.4, η denotes the learning rate. However, because of the scaling with the inverse square root of the averaged second moments $\hat{v}_t$, it adapts to each parameter individually. Intuitively, ADAM behaves like a heavy ball with friction. Because the first momentum, i.e. the running average of the gradient $\hat{m}_t$ defines the step direction and the unscaled step size, it tends to skip narrow, local minima of the loss function and prefers flat ones instead.

3.2.2 Backpropagation and Automatic Differentiation

A key element in the update step of the minimization algorithms presented above is the computation of the gradient with respect to the network parameters. To derive the gradients, a successive application of the chain rule is necessary. Similar to the evaluation of the network in equation 3.2, a recursive pattern reveals. If expressed as an algorithm, it is known as the backpropagation algorithm [Wer75]. Including the weight update during a single cycle of an optimization algorithm, the method—in contrast to the forward pass—is called the *backward pass*.

Implementing the backpropagation by hand can be tedious, error prone and requires a reimplementation for every other network architecture, e.g. when replacing the activation function. To overcome this inconvenience, many popular machine-learning frameworks, e.g. TensorFlow [ABC+16] and PyTorch [PGM+19] introduced a technique called *automatic differentiation*. It allows to instruct the framework to track computations that involve specified variables e.g. the network parameters. Behind the scenes, the framework then generates, what is called a *computation graph*. The leafs of this graph comprise the variables and the nodes are operations. By applying the chain rule successively, the framework can then compute the gradients by itself, provided that the derivation of each node is well-defined. If this is not the case, it can be manually proved by the user. However, typically this is the case for all common operations and activation functions e.g. tanh.

3.3 Physics Informed Neural Networks

Before ANNs were applied as physics-informed learning machines, early approaches employed Gaussian process regression [Ras04] for linear [Owh15; RPK17a; RPK17b], and non-linear PDEs [RK18; RPK18]. However, in the non-linear case, these methods showed severe limitations concerning the accuracy and approximation capacity of the resulting models.

The central concept of PINNs is to include prior information, given by the PDE, during the training procedure of the ANN-based model. This is achieved via a specially designed loss function that penalizes the network's deviation from the rules described by the PDE. Raissi et al. [RPK19] originally described his framework for two types of problems: data-driven solution and data-driven discovery. The following introduces the first problem, which concerns inferring a model for the latent solution of the PDE.

Consider a general nonlinear PDE with its boundary and initial condition:

$$\begin{aligned} \mathcal{N}_{x,t}[u(x,t)] &= 0, \quad x \in \Omega, t \in [0,T] \\ \mathcal{B}_x[u(x,t)] &= 0, \quad x \in \partial\Omega, t \in [0,T] \\ u(x,0) &= u_0(x), \quad x \in \partial\Omega \end{aligned} \tag{3.6}$$

Here, x denotes the spatial component, with $\Omega \subset \mathbb{R}^d$ and t is time. The latent solution function of the PDE is $u(x,t)$ and $\mathcal{N}_{x,t}$ is a spatio-temporal, non-linear differential operator applied on u. In contrast to the original work by Raissi et al., who only consider Dirichlet type boundary conditions, we introduce a spatial differenial $\mathcal{B}_x$ operator to allow also Neumann or higher-order boundary conditions. The function $u_0(x)$ imposes the initial conditions. If we approximate the solution function u by an artificial neural network (ANN) $\tilde{u}_\theta$ with parameters θ and define the right-hand side of the PDE by

$$f_\theta(x,t) := \mathcal{N}_{x,t}[\tilde{u}_\theta(x,t)], \tag{3.7}$$

then $f_\theta(x,t)$ is called a *physics-informed neural network (PINN)*. It is worth mentioning that it shares the same parameters as $\tilde{u}_\theta$. To ensure that the approximating solution $\tilde{u}_\theta$ fulfills the PDE and its initial and boundary conditions imposed by equation 3.6, the accumulation of three distinct loss functions $\Psi_f(\theta)$, $\Psi_0(\theta)$ and $\Psi_b(\theta)$ form the loss function $\Psi(\theta)$ during the training phase:

$$\Psi(\theta) \;=\; \omega_f \Psi_f(\theta) \;+\; \omega_0 \Psi_0(\theta) \;+\; \omega_b \Psi_b(\theta) \tag{3.8}$$

Here, $\Psi_f(\theta)$ denotes the *PDE loss*, which penalizes the deviation of the PINN from zero. The *initial loss* Ψ_0 penalizes deviations from the initial condition function at $t = 0$. The *boundary conditions* are enforced by the boundary loss Ψ_b. The weights ω_f, ω_0, and ω_b scale the losses and therefore scale their gradients during the backward pass. That, in turn, results in an individual learning speed for each of the three requirements necessary for $\hat{u}$ to approximate the latent solution of the PDE. To achieve a reasonable convergence speed of the minimization algorithm, the weights must be

fine-tuned. However, recent methods [WTP20; WYP20] dynamically adapt the weights in each optimization step and work well out of the box. Specifically, the method by Wang et al. [WTP20] is used in this work and will be further explained in Section 4.2.1. The losses have the form:

$$\Psi_f = \frac{1}{N_f} \sum_{i=1}^{N_f} \psi_f \left(f(x_i^f, t_i^f) \right) \tag{3.9}$$

$$\Psi_0 = \frac{1}{N_0} \sum_{i=1}^{N_0} \psi_0 \left(\tilde{u}(x_i^0, 0), u_i^0 \right) \tag{3.10}$$

$$\Psi_b = \frac{1}{N_b} \sum_{i=1}^{N_b} \psi_b \left(\mathcal{B} \left[\tilde{u}(x_i^b, t_i^b) \right] \right) \tag{3.11}$$

Each loss function requires a distinct set of training data. Here, the spatio-temporal points $\{(x_i^f, t_i^f)\}_{i=1}^{N_f}$ are called *collocation points*. The labeled set $\{(x_i^0, u_i^0)\}_{i=1}^{N_0}$ denotes the *initial points* which are sampled from the initial condition i.e. $u_i^0 = u_0(x_i^0)$. Similarly, the *boundary points* $\{(x_i^b, t_i^b)\}_{i=1}^{N_b}$ are sampled along the spatial boundary for various times i.e. $x_i^b \in \partial\Omega$. Note that no labels define the outcome at the boundary in the case of Dirichlet boundary conditions. Here, the required value is wrapped as the outcome of a function inside the non-linear differential operator $\mathcal{B}_x$. The collocation points can be sampled randomly across the spatio-temporal domain. However an improved sampling strategy, e.g. by concentrating on regions with higher error, can improve the convergence during the training phase [WZ20] Especially, when considering higher dimensional domains—similar to the discretization in classical numerical solvers—improved strategies are essential to overcome the curse of dimensionality [Bel66].

4 Methods

This chapter presents the methods used to solve the SWE within the framework of PINN. The first section considers the fundamental model and training assumptions, including implementation details of the network architecture and the PINN, as well as sampling strategies for the training and validation data. The following section presents optimization techniques investigated to speed up the training process and improve accuracy.

4.1 Model and Training assumptions

4.1.1 Network and Training

As mentioned earlier, all implementations were made in Python, and the machine learning framework PyTorch was chosen for defining and training ANNs. PyTorch provides high-level access to common deep-learning architectures and techniques. This also includes the computation of derivatives with automatic differentiation, provided by the `torch.autograd` module. All the training in this work was performed on an Nvidia GTX 970 with 4GB of memory. In all scenarios, a fully connected feed-forward network serves as the baseline architecture for other improvements. Similar to the approach presented by Raissi et al. [RYK20] weight normalization is applied to improve the training speed. It is a re-parameterization of the weight vectors in a neural network that decouples the length of the weight vectors from their direction, and an alternative to batch normalization. In all cases, the inputs of the network, i.e. spatio-temporal points, are normalized by subtracting the mean and dividing by the standard deviation, which leads to a mean of zero and a standard deviation of one. Unless it is mentioned otherwise, the method of Glorot et al. [GB10] serves as the default initialization scheme of the network weights and biases.

As already described for the general case in Section 3.3, to train the model, all three losses require individual datasets. The choice of the dataset heavily impacts the training and can analogously be viewed as the discretization scheme in classical numerical approaches such as finite differences. This work considers random sampling based on the latin hypercube sampling (LHS) strategy [Ima99], which ensures that no data point is sampled twice. As the SWE are hyperbolic type of PDEs, it has well-posed initial value problems which are only concerned in this work. Therefore no boundary conditions are employed and only collocation and initial points are sampled. To concretize this, the N_f collocation points $\{(x_i^f, t_i^f)\}_{i=1}^{N_f}$ are sampled in the spatio-temporal domain $\Omega \times [0, T]$. The N_0 initial points $\{(x_i^0, 0)\}_{i=1}^{N_0}$ and the labels $\{u_0(x_i^0)\}_{i=1}^{N_0}$ are sampled from the inital condition, along the spatial domain Ω. For all trained PINNs I used $N_f = 20\text{K}$ collocation and $N_0 = 1000$ initial points. Obviously, random sampling is not the most economic solution concerning memory usage and training speed. However, as only 1D problems are concerned in this work the datasets easily fit on low end graphics cards or can be trained entirely on the cpu. Recent work suggest adaptive sampling strategies which can improve accuracy and convergence [WZ20]. They present a *time-adaptive sampling* approach, where the sampling space of the collocation points is gradually increased along the temporal direction during training, based on the value of the PDE loss. One test case presented in Section 5.2 compares this strategy with the one where all collocation points are used in each training step. Figure 4.1 illustrates the two methods.

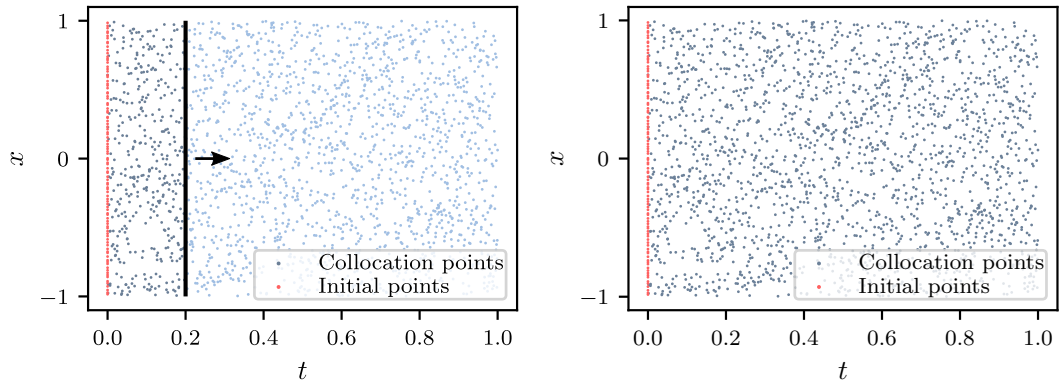

Figure 4.1: A training dataset with 2000 collocation and 100 initial points randomly sampled with the LHS method. The left image illustrates the time-adaptive sampling where the sampling space of the collocation points is gradually increased along the temporal direction; here the limiting timestep is at $t = 0.2$. The right image represents the whole training dataset as it is used in classical random sampling.

As an evaluation metric I compare a model's prediction with the outcome of a Pyclaw simulation at 10K points randomly sampled from the discretization. For each of the predicted quantities, i.e. water height and velocity (or momentum), the relative error in the L^2 norm was computed. The relative L^2 error between a prediction f and ground truth g at points $\{x_i\}_{i=1}^N$ is defined as

$$\mathcal{E}(f,g) := \left(\frac{1}{N}\sum_{i=1}^{N}\left[f(x_i)-g(x_i)\right]^2\right) \Big/ \left(\frac{1}{N}\sum_{i=1}^{N}\left[g(x_i)-\frac{1}{N}\sum_{i=1}^{N}g(x_i)\right]^2\right), \tag{4.1}$$

and as a single metric I defined the mean of all relative errors for k quantities $\mathcal{E}_1, \ldots, \mathcal{E}_k$

$$\mathcal{E}_{\text{tot}} := \frac{\mathcal{E}_1 + \cdots + \mathcal{E}_k}{k}, \tag{4.2}$$

as the total *validation error*.

4.1.2 PINN for the 1D Shallow Water Equations

When concerning the SWE, it is not obvious, which of the two forms, non-conservative or conservative, should be chosen to define the PINN. Here, no discretized derivatives need to be computed that would influence the choice. However, although they are mathematically equivalent, each representation has advantages and disadvantages which are described in the following.

Conservative form

Consider the conservative form of the 1D SWE as defined in the Equations 2.1 and **??**. A first attempt would be to define the ANN such that it takes the spatio-temporal coordinate $(x,t) \in \mathbb{R} \times [0,T]$ as input and returns the prediction of water height and momentum $(\tilde{h}, \tilde{hu})$ as output. This seems to be a good choice at first glance, but a drawback shows when defining the PINN. Because of the term $\frac{\partial hu^2}{\partial x}$ it is necessary to multiply the velocity u with the momentum hu. As the velocity is not part of the prediction, it requires to divide the momentum by the height, i.e. $\tilde{u} := \frac{\tilde{hu}}{\tilde{h}}$. This leads to the following PINN:

$$f_\theta^{\mathrm{C}}(x,t) := \begin{pmatrix} \frac{\partial \tilde{h}}{\partial t} + \frac{\partial \tilde{hu}}{\partial x} \\ \frac{\partial \tilde{hu}}{\partial t} + \frac{\partial (\tilde{hu})^2/\tilde{h} + g\tilde{h}^2/2}{\partial x} + g\tilde{h}\frac{\partial b}{\partial x} \end{pmatrix} \tag{4.3}$$

This is not optimal, since the prediction returned by the ANN might not obey the rules of the PDE during the training, it can take values of zero or close to zero that lead to crashes or high loss values which slow down training or inhibit learning at all. This is particularly problematic at the beginning of the training, where the initialization of the parameters determine the output of the ANN. To overcome this issue, an initialization scheme that prevents zero or close to zero output is essential. Additionally a regularization term, that penalizes small absolute values of the depth prediction $\tilde{h}$, e.g. $\frac{1}{|\tilde{h}|+\epsilon}$ can be introduced as an additional loss. Observations show, that using the parameter initialization presented by Xavier et al. [GB10] with the bias set to 1.01 (which prevents a zero network output), already yields reasonable performance.

A simpler approach is to directly output the velocity prediction $\tilde{u}$ instead of the momentum prediction $\tilde{hu}$, resulting in the following PINN

$$f_\theta^{NC}(x,t) := \begin{pmatrix} \frac{\partial \tilde{h}}{\partial t} + \frac{\partial \tilde{h}\tilde{u}}{\partial x} \\ \frac{\partial \tilde{h}\tilde{u}}{\partial t} + \frac{\partial \tilde{h}\tilde{u}^2 + g\tilde{h}^2/2}{\partial x} + g\tilde{h}\frac{\partial b}{\partial x} \end{pmatrix}. \tag{4.4}$$

However, this solution does not differ from the non-conservative form described later.

A benefit from considering the momentum directly is that by a smart trick one can get rid of the first equation and obtain mass conservation implicitly. The underlying concept has been presented in other works to acquire a divergence-free model of the Navier-Stokes equations [KAT+19; RPK19]. To my best knowledge, this is the first time anyone applied it to the SWE. The method exploits the fact, that the curl of the gradient of a scalar-valued function is always zero. Consider the scalar-valued function $\phi : \Omega \subset \mathbb{R} \times [0,T] \to \mathbb{R}$ where $(x,t) \mapsto \phi(x,t)$ and ϕ is at least twice continuously differentiable, i.e. $\phi \in C^2$, then the following holds:

$$\operatorname{curl} \nabla\phi(x,t) = \nabla \times \begin{pmatrix} \frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial t} \end{pmatrix} = \frac{\partial^2 \phi}{\partial t \partial x} - \frac{\partial^2 \phi}{\partial x \partial t} = 0 \tag{4.5}$$

By choosing $\tilde{h} := -\frac{\partial \phi}{\partial x}$ and $\tilde{hu} := \frac{\partial \phi}{\partial t}$, then

$$\text{curl}\, \nabla \phi(x,t) = \frac{\partial \tilde{hu}}{\partial x} + \frac{\partial \tilde{h}}{\partial t} = 0, \tag{4.6}$$

which resembles the mass conservation defined in Equation **??**. Transferring this to the model, means defining the ANN as the scalar-valued function ϕ, and thus a surface in the spatio-temporal domain is learned whose gradient reflected across the time axis is an approximation to the solution function of the shallow water equations. To train the network only the momentum equation must be enforced via the PDE-loss and therefore the PINN has the following form:

$$f_\theta^\phi(x,t) := \frac{\partial \tilde{hu}}{\partial t} + \frac{\partial (\tilde{hu})^2/\tilde{h} + g\tilde{h}^2/2}{\partial x} + g\tilde{h}\frac{\partial b}{\partial x}, \quad \text{with} \quad \begin{pmatrix} \tilde{h} \\ \tilde{hu} \end{pmatrix} := \begin{pmatrix} -\frac{\partial \phi}{\partial x} \\ \frac{\partial \phi}{\partial t} \end{pmatrix} \tag{4.7}$$

It is worth mentioning, that the requirement of being twice continuously differentiable is fulfilled by the ANN exactly when the activation function is twice continuously differentiable. This does not collide with the activation functions used in this work as they are infinitely continuously differentiable, i.e. $\{\tanh, \sin\} \subset C^\infty$. The drawbacks of this approach are similar to that of Equation 4.3. Here, the negative partial derivative of the surface with respect to the spatial direction is not allowed to be zero. However finding an appropriate initialization scheme, that (1) yields either a strictly monotonically increasing or decreasing surface along the x direction and (2) preserves good training performance, is not trivial, as it depends on the activation function and must be further investigated. Luckily, for the tanh activation function the initialization scheme by Xavier et al. [GB10] with the bias set to 0 leads to acceptable performance. For the sin activation function, however, this scheme does not work well and produces high loss values directly at the beginning, which lead to slow convergence. It should also be mentioned that the training time is larger compared to the other PINNs in this work (I measured an increase of about 1.5 concerning a model with 20000 collocation and 1000 initial points and architecture with 5 hidden layers à 20 neurons). This mainly results from the computation of the second derivatives, which requires populating a computation graph that has double the size compared to that of a first derivative.

Non-Conservative form

Similarly to the previous section, the PINN for the non-conservative form of the SWE can be defined by:

$$f_\theta^{\text{NC}}(x,t) := \begin{pmatrix} \frac{\partial \tilde{h}}{\partial t} + \tilde{h}\frac{\partial \tilde{u}}{\partial x} + \tilde{u}\frac{\partial \tilde{h}}{\partial x} \\ \frac{\partial \tilde{u}}{\partial t} + \tilde{u}\frac{\partial \tilde{u}}{\partial x} + g\left(\frac{\partial \tilde{h}}{\partial x} + \frac{\partial b}{\partial x}\right) \end{pmatrix}. \tag{4.8}$$

Equivalent to Equation 4.4 this approach considers the velocity directly and therefore does not suffer from division by zero issue. The downside of this approach is, that one can not eliminate the mass conservation equation and needs to penalize non-fulfillment.

Algorithm 4.1 Learning rate annealing (LRA) for physics-informed neural networks adapted from [WTP20]

Consider the accumulated loss function similar to that described in Equation 3.8 but with an arbitrary number of components:

$$\Psi(\theta) = \Psi_f(\theta) + \sum_i^M \omega_i \Psi_i(\theta),$$

where $\Psi_f(\theta)$ denotes the PDE loss, the $\Psi_i(\theta)$ are other losses (e.g. initial or boundary conditions) and $\omega_i = 1, i = 1, \ldots, M$ are weights to balance the interplay between the different loss terms. In each training epoch, after computing the gradients and before performing the t-th optimizer step, update the weights as:

for $i = 1, \ldots, M$ **do**

$$\hat{\omega}_i \leftarrow \frac{\max_\theta \{|\nabla \Psi_f(\theta_{t-1})|\}}{\overline{|\nabla \Psi_i(\theta_{t-1})|}}$$

$$\omega_i \leftarrow (1-\alpha)\omega_i + \alpha \hat{\omega}_i$$

end for

Here $\overline{|\nabla \Psi_i(\theta_{t-1})|}$ denotes the mean of $|\nabla \Psi_i(\theta_{t-1})|$ with respect to all parameters θ. The hyper-parameter α is set to its recommended value of 0.9 throughout this work.

4.2 Optimization Methods

This section explains the optimizations methods which were examined to increase training speed and the accuracy of the learned model.

4.2.1 Learning Rate Annealing

Wang et al. [WTP20] considered training the Helmholtz equation with a PINN and observed that the error along the boundary is particularly high. By inspecting the gradients of each individual loss function, they found that the gradient of the PDE loss, with respect to the parameters, is larger than that of the boundary loss. Since the total loss Ψ is the composition of the three losses, as defined in Equation 3.8, the gradient of the total loss is the sum of the gradients of the individual losses. Therefore, the influence of the boundary was smaller during the parameter update (see Section 3.2.1). Which ultimately biased the network towards neglecting the contribution of the boundary loss. To compensate for the effect, they provide a heuristic weighting that leads to more balanced gradients during the backpropagation. The *learning rate annealing (LRA)* method is defined in Algorithm 4.1. As the algorithm describes, the method is not limited to boundary conditions but considers all additionally applied loss terms individually by dynamically scaling them according to the performance of the PDE loss. The method draws its motivation from the ADAM optimizer [KB17] and uses statistics of the gradients to update the weights in a moving average like fashion. The benefit of the method is that it eliminates the trial-and-error procedure of finding weights with good convergence, which is an extremely tedious task, especially when the number of losses increases.

4.2.2 Sinusoidal Representation Networks

The choice of the activation function can make a high impact on training performance. Sitzmann et al. [SMB+20] present *sinusoidal representation networks (SIRENs)* which correspond to fully connected, feed-forward networks with a sinus activation function. They applied their network to a wide variety of problems, ranging from inferring a solution to the Helmholtz and wave equation with PINNs up to representing shapes with signed distance functions. The results show that the sine activation outperforms other architectures that use classical activations such as the hyperbolic tangent tanh or ReLU in convergence speed and accuracy. For example, they achieved an improvement of over one order of magnitude for the wave equation compared to the baseline approach using tanh. In the supplementary part of their work, they describe additional details on the exact implementation and reveal further optimizations to the original version of the network. They suggest scaling the network weights by a factor κ to increase the convergence speed further. The reason for this is that during backpropagation, the factor remains and scales the gradient of the loss with respect to a weight, which consequently leads to an increased update step during optimization. Including this optimization, the SIREN has the following form:

$$y = f_\theta(x) = \sin(\kappa W^k \dots \sin(\kappa W^2 \sigma(W^1 x + b^1) + b^2) + \dots) \tag{4.9}$$

A specific initialization scheme is necessary to train the network successfully. It should avoid that the output of the last layer depends on the number of layers. They propose to sample the weights W^l uniformly, such that

$$\begin{aligned} W^1 &\sim \mathcal{U}\left(-\sqrt{\frac{6}{\text{fan\_in}_1}}, \sqrt{\frac{6}{\text{fan\_in}_1}}\right), \\ W^l &\sim \mathcal{U}\left(-\sqrt{\frac{6}{\kappa^2 \text{fan\_in}_l}}, \sqrt{\frac{6}{\kappa^2 \text{fan\_in}_l}}\right), \quad \text{for } l > 1. \end{aligned} \tag{4.10}$$

Here, l denotes the layer and the term fan_in_l describes the number of neurons in the previous layer $l-1$, i.e. the number of columns in W^l. The different sampling of weights in the first layer introduces higher frequencies at the beginning of the training, that further increases the convergence. While they recommend setting $\kappa = 30$, this produced poor results. After some evaluation a value of 3.5 turned out to be much more appropriate for the scenarios considered in this work.

4.2.3 Adaptive Activation

In the context of physics-informed neural networks, Jagtap et al. [JKK19] present a procedure, which speeds up the training process by introducing scaling factors inside the activation function, similar to the parameter κ described in the section above. However, the scaling factors described in their work are part of the network's parameters and are trained during the optimization. They propose the introduction of these parameters at three different levels of granularity—a single

parameter for the entire network, layer-wise, and neuron-wise. This work examines the layer-wise and neuron-wise approach, as they showed most prospects considering the experiments presented in their work. To further increase the training speed, they also propose an additional loss term that guides the optimization of the parameters. In a theoretical analysis, they showed that the method is non-attractive to sub-optimal local minima during gradient descent. The following describes the methods.

Consider the feed-forward neural network as defined in Equation 3.2. The layer-wise scaled activation functions, denoted by *layer-wise locally adaptive activation functions (L-LAAF)*, which are introduced in the hidden layers of a network with width p depth k are defined:

$$\sigma(g\lambda^l a^{l-1}), \quad \text{with } l = 1, \ldots, k-1 \tag{4.11}$$

Similarly, activation functions on the neuron level called *neuron-wise locally adaptive activation functions (N-LAAF)* are defined as:

$$\sigma(g\lambda_i^{l-1} a_i^{l-1}), \quad \text{with } l = 1, \ldots, k-1, \text{ and } i = 1, \ldots, p \tag{4.12}$$

Here, the λ's are new network parameters that are trained during the optimization step. The L-LAAF introduces $k-1$ and N-LAAF introduces $p(k-1)$ additional parameters. The global scaling factor g is a hyper-parameter that is problem dependent and can decrease the performance if set too high. After some experiments, it showed that $g = 1$ is an appropriate value for the L-LAAF method and $g = 5$ for the N-LAAF method. The parameters should be initialized such that $\lambda = \frac{1}{g} \forall \lambda$. In both cases, the primary motivation of the scaling factors is to increase the slope of the activation function, which results in non-vanishing gradients and faster training of the network. To profit from faster convergence during the beginning of the training, Jagtap et al. introduce the slope recovery loss term that penalizes small values of the new parameters and therefore speeds the slope increase. It is defined as

$$\mathcal{S}(\lambda) := \begin{cases} \frac{1}{\frac{1}{k-1}\sum_{l=1}^{k-1} \exp(\lambda^l)} & \text{for L-LAAF} \\ \frac{1}{\frac{1}{k-1}\sum_{l=1}^{k-1} \exp\left(\frac{\sum_{i=1}^{p} \lambda_i^l}{p}\right)} & \text{for N-LAAF} \end{cases} \tag{4.13}$$

and adds to the total loss similar to the other losses defined in Equation 3.8.

4.2.4 Attention Mechanism

Besides the LRA method, another way to increase the accuracy of the network's prediction was presented recently by McClenny et al. [MB20]. Their approach focuses on "stiff" PDEs, which are characterized by sharp spatio-temporal transitions. Solving these types of problems with the original method by Raissi et al. [RPK19] can lead to poor results and typically requires an unreasonably large

number of collocation points. To overcome this problem, McClenny et al. suggest an attention-based approach that weights each point individually and dynamically draws attention to regions that yield lower performance during training. Attention-mechanism is a well-known technique in deep learning and is applied in transformer networks for language translation [VSP+17] and object detection [LLZC20]. The following describes the approach in more detail.

As a foundation, consider the three loss terms defined in Equations 3.9, 3.10, and 3.11. Introducing the trainable *self-adaption weights* $\xi^f := \left\{\xi_i^f\right\}_{i=1}^{N_f}$, $\xi^0 := \left\{\xi_i^0\right\}_{i=1}^{N_0}$, and $\xi^b := \left\{\xi_i^b\right\}_{i=1}^{N_b}$ for each of the collocation, initial, and boundary points allows weighting each point-wise loss individually. By specializing on the MSE loss, we can adapt the three loss terms as:

$$\Psi_f(\theta, \xi^f) = \frac{1}{N_f} \sum_{i=1}^{N_f} \left[\xi_i^f \, f(x_i^f, t_i^f)\right]^2 \tag{4.14}$$

$$\Psi_0(\theta, \xi^0) = \frac{1}{N_0} \sum_{i=1}^{N_0} \left[\xi_i^0 \left(\tilde{u}(x_i^0, 0) - u_i^0\right)\right]^2 \tag{4.15}$$

$$\Psi_b(\theta, \xi^b) = \frac{1}{N_b} \sum_{i=1}^{N_b} \left[\xi_i^b \mathcal{B}\left[\tilde{u}(x_i^b, t_i^b)\right]\right]^2 \tag{4.16}$$

That leads to the composite loss function:

$$\Psi(\theta, \xi^f, \xi^0, \xi^b) := \Psi_f(\theta, \xi^f) + \Psi_0(\theta, \xi^0) + \Psi_b(\theta, \xi^b) \tag{4.17}$$

The key feature of the approach is to minimize the composite loss with respect to the network parameters θ, while maximizing it with respect to the self-adaption weights ξ^f, ξ^0, ξ^b. Consequently, the training procedure seeks a saddle point

$$\min_{\theta} \max_{\xi^f, \xi^0, \xi^b} \Psi(\theta, \xi^f, \xi^0, \xi^b). \tag{4.18}$$

Because of the maximization, those weights at points that have a higher error are preferably increased. This leads to an even higher loss at those points which the minimization tries to decrease by adjusting the parameters of the network. Ultimately, this counterplay directs the focus at locations with a lower performance during the minimization, which speeds up the overall convergence. The authors do not directly provide a recommended initialization scheme but prove that the weights monotonically increase as long as initialized positively. In the investigated examples, they sample the weights from a uniform distribution, i.e. $\xi^0 \sim \mathcal{U}[0, 100]$ and $\xi^f \sim \mathcal{U}[0, 1]$ which is also applied in this case, however, decreasing the standard deviation for the initial points to 10. Note that one can still use a hard-coded weighting of the individual loss as defined in equation 3.8 to apply a general preference on the three losses. From an implementation point of view, the training goal can be achieved conveniently with any gradient descent based optimizer by simply flipping the sign of

the update of the self-adaption weights. In their work, they describe a two-step training approach, were during a pre-training phase, the network trains along with the weights for a certain number of epochs using the ADAM optimizer. Afterward, the weights are kept fixed, and the network trains for another number of epochs using the L-BFGS optimizer [LN89]. I tried the same procedure, however, in the second phase, the optimizer tends to get stuck in local minima or the total loss explodes, and is highly sensitive to the applied learning rate. Even if in some cases it converged quicker than ADAM with the learning rate suggested in their work, for a fair comparison to the other optimization methods, I also used ADAM in the second phase with fixed weights.

5 Results

This chapter discusses the results achieved after solving three different SWE scenarios with the PINN framework as described in the previous chapter. The first section presents a detailed analysis of the architecture choice and performance of the PINN definitions considering a simple scenario without bathymetry and gentle steepening of the wavefront. Findings of these investigations form the basis for the following scenarios. The next section investigates the idealized dam break scenario that shows an increase in complexity as it contains non-continuous initial conditions and shocks that propagate through space as time passes. The last section considers a depth perturbation with bathymetry where steepening of the wavefront emerges stronger than in the first scenario. A strong focus of this chapter lies on the performance of the optimization strategies. In each scenario the optimization strategies are compared with each other after training 50K epochs, where the learning rate is set fixed to 0.01. Throughout the experiments the learning rate showed to be appropriate and allows for a fair comparison between the methods. Although it is not necessarily practical, for each scenario, I present a summary of the solutions with smallest validation error achieved with the different optimization methods epochs. Additionally, the best model overall is illustrated to get an impression of which result one can theoretically achieve. However, I will remark how realistic it is to achieve such values when no validation data is present.

5.1 Scenario 1 — Small Depth Perturbation

This section considers a depth perturbation, similar to the one illustrated in Figure 2.2 but with a smaller initial height and decreased gravity. The initial conditions of the initial value problem are defined as

$$\begin{aligned} h(x,0) &= 0.2 \cdot \exp\left(\frac{-x^2}{0.4}\right) + 0.8, \\ hu(x,0) &= 0, \end{aligned}$$

where the considered domain is limited to $x \in [-1, 1]$ and $t \in [0, 1]$. To delay the steeping of the wavefront in this area, the propagation speed of the wave is reduced by setting the gravity $g = 2$. Figure 5.1 illustrates the propagation of the wave and its horizontal momentum at three timesteps extracted from a Pyclaw [KMA+12] simulation.

5.1.1 Architecture Choice

To get a feel of how many layers and neurons per layer are needed to achieve an acceptable solution accuracy, I trained different network architectures, all based on the non-conservative PINN defined in 4.8, to approximate the simulation results. The test is performed for both, the hyperbolic tangent and sine activation function. I did not choose the scalar-output model intentionally, as it does not perform well on the SIREN. Although it has one neuron less in the output layer, the difference in approximation quality should vanish when the number of neurons and layers increase. As already described, the feed-forward network of the non-conservative form outputs the water height h and velocity u. The total loss function used in this case only penalizes deviations of the network's output from the simulation results. Therefore, 20K distinct points are sampled randomly from

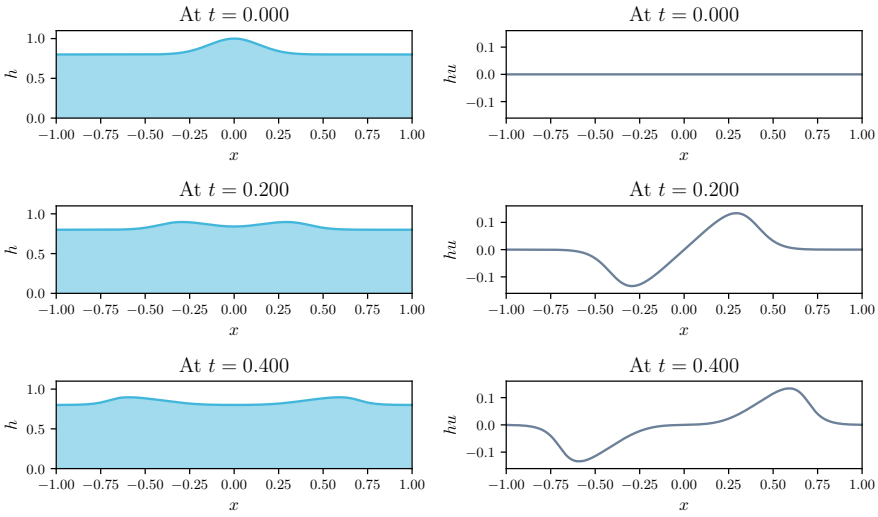

Figure 5.1: Scenario 1 — Three timesteps extracted from a Pyclaw [KMA+12] simulation with a spatial discretization consisting of 500 cells, and gravity g set to 2.0. The left column shows the water depth h and the right column the momentum hu.

the simulation solution. I divided the momentum output of the simulation by the height to get the velocity needed for the training data. Note, that the definition of the PINN does not play an essential role here, as the fully connected feedforward-network is almost the same in all three cases. Only the scalar-output network defined in 4.7 has one neuron less in the output layer, which does not make a great difference as experiments showed (e.g. 2.86×10^{-3} for a depth of 5 and width of 20). It should also be mentioned, that this investigation is just for studying purposes and can not be applied when the solution is unknown. In such cases, one can consider approaches such as Bayesian optimization [SLA12] to tune the network architecture. To make the architecture types comparable, here the SIREN boosting factor κ was set to 1.0. Tables 5.1 and 5.2 summarize the best validation losses achieved after training networks with different widths and depths for 10K epochs and a learning rate of 0.01. The former Table corresponds to networks with hyperbolic tangent activation function, the latter to ones with sine activation function.

The results show, that apart from a few exceptions, the performance of the network grows with the number of layers and neurons per layer. The exceptions should vanish when the number of epochs reach a point where the capacities of all networks are fully exploited and differences in the training speed that result from differences in the landscape of the loss function do not play a role anymore. I decided to stick with the network of depth 5 and width 20 with 1280 weights and 82 biases, as it performed well and seemed to have more than sufficient approximation capacity with an acceptable amount of computing time, i.e. here $\approx 8 \times 10^{-3}$ seconds for an epoch. The sufficient approximation capacity turns out to be valid, as we will see in the following examinations.

Depth \ Width	5	10	20	50
2	1.723×10^{-1}	3.945×10^{-2}	3.079×10^{-2}	5.343×10^{-2}
3	1.169×10^{-2}	5.455×10^{-3}	4.287×10^{-3}	1.213×10^{-2}
4	2.010×10^{-2}	3.104×10^{-2}	8.825×10^{-3}	2.786×10^{-3}
5	9.640×10^{-3}	1.109×10^{-2}	1.832×10^{-3}	6.888×10^{-4}

Table 5.1: Scenario 1 — Relative L^2 error to simulation data for different network architectures with **hyperbolic tangent** activation function. Each network was trained to approximate 20K distinct points randomly sampled from the simulation data. The values correspond to the best approximation achieved after 10K epochs.

Depth \ Width	5	10	20	50
2	3.659×10^{-1}	9.823×10^{-2}	1.679×10^{-1}	4.751×10^{-2}
3	5.609×10^{-2}	1.333×10^{-2}	5.548×10^{-3}	1.964×10^{-3}
4	1.809×10^{-2}	4.891×10^{-3}	3.952×10^{-3}	1.520×10^{-3}
5	6.871×10^{-3}	2.258×10^{-3}	3.170×10^{-3}	7.484×10^{-4}

Table 5.2: Scenario 1 — Relative L^2 error to simulation data for different network architectures with **sine** activation function. Each network was trained to approximate 20K distinct points randomly sampled from the simulation data. The values correspond to the best approximation achieved after 10K epochs.

5.1.2 PINN Choice

As mentioned earlier, it is not straightforward, which of the PINNs defined in section 4.1.2 to choose. It seems obvious to prefer the non-conservative form f_θ^{NC}, as it is the only one where the training process can not crash due to a division by zero. However, it is still interesting to see how each term performs when solving the initial value problem. For a better understanding, I trained the different forms for 50K epochs using the network architecture described in the previous section. To allow a fair comparison to the scalar-output approach, the hyperbolic tangent activation function was used. Also, utilizing the LRA method provides an unbiased weighting scheme for the initial and PDE losses. Additionally, to allow a better comparison between the non-conservative and conservative PINN, I trained the non-conservative form twice, once with a bias of 0 and another time, with bias set to 1.01. The learning rate is was set fixed to 0.01. Table 5.3 summarizes metrics of the models that produced the lowest validation error after training 50K epochs. Additionally, Figure 5.2 illustrates the evolution of various training and validation metrics.

It shows that the scalar-valued PINN performed worst with a difference of more than one and a half orders of magnitude to the non-conservative form with same bias. Clearly, the initialization scheme is not optimal as one can see when comparing the two non-conservative models. But the fact that it performed similarly during the simulation approximation leads to the guess that due to the

PINN	total err.	h err.	hu err.	mass eq.	mom. eq.	epoch
f_θ^{C}	2.8×10^{-4}	3.2×10^{-4}	2.4×10^{-4}	2.5×10^{-6}	2.7×10^{-6}	45,837
f_θ^{ϕ}	5.5×10^{-1}	6.7×10^{-1}	4.4×10^{-1}	0.0	4.1×10^{-4}	5,417
f_θ^{NC}, $b = 0$	2.1×10^{-3}	2.3×10^{-3}	2.0×10^{-3}	2.8×10^{-5}	1.4×10^{-5}	49,729
f_θ^{NC}, $b = 1.01$	1.2×10^{-4}	1.3×10^{-4}	1.1×10^{-4}	3.1×10^{-6}	2.3×10^{-6}	49,530

Table 5.3: Scenario 1 — Performances of different PINNs forms, each row corresponds to the network with lowest total validation error achieved after training 50K epochs.

deep architecture resulting from the second derivatives, the training speed suffers from vanishing gradients. One way to address this is to use a different activation function that does not scale down the gradient of activated neurons (the derivative of tanh converges to 0 for both positive and negative inputs) and is at least twice continuously differentiable e.g. the Swish function. Also the adaptive activation methods described in Section 4.2.3 showed a speed-up of the training process.

Comparing the conservative network in the first row of Figure 5.2 with both non-conservative cases, clearly shows that the increased bias is the main reason for the improved performance, and only comparing with the 0 bias case would have lead to a wrong conclusion. Comparing both cases where the bias is set to 1.01, one can see that the conservative form converges faster at the beginning, but the validation errors equalize near the end. That leads to the conclusion that applying the non-conservative form does not require making sacrifices (also the same was true for more complex scenarios that face discontinuities), and therefore, the following approaches only concern this form. Also, Table 5.3 shows that mass conservation is fulfilled to a satisfying amount and does not justify employing the scalar-output based approach.

5.1.3 Optimizations

When comparing the results of the two non-conservative forms in Table 5.3 one can see that the initialization scheme has a significant impact on the convergence speed, which should be examined further. For example, a meta-learning based approach similar to the one Dauphin et al. [DS19] could yield appropriate initializations for the weights and biases. However, in this work, the main focus was to increase the accuracy and training speed of problems with stronger steepening wavefronts and higher frequency components in the solution. Therefore optimization strategies were preferred that showed prospects in this direction.

To compare the different optimization methods, I considered the same network as in the previous section and the non-conservative form due to its proven stability as described above. Additionally, 20K collocation and 1K initial points were sampled randomly with the LHS method and used in all training epochs. Also, similar to the previous test the learning rate is set fixed to 0.01. Figure 5.3 depicts the evolution of the training and validation metrics for all of the trained 50K epochs, and Table 5.4 summarizes the results as in the previous case. Here, the LRA method, also used in the previous section serves as the baseline and was used in all cases except for the attention-based approach (which learns a weighting of the loss terms on a pointwise level). For a more comfortable comparison, the result of the plain LRA approach, which was also shown in the PINN comparison in Figure 5.2 and Table 5.3 is plotted and listed again.

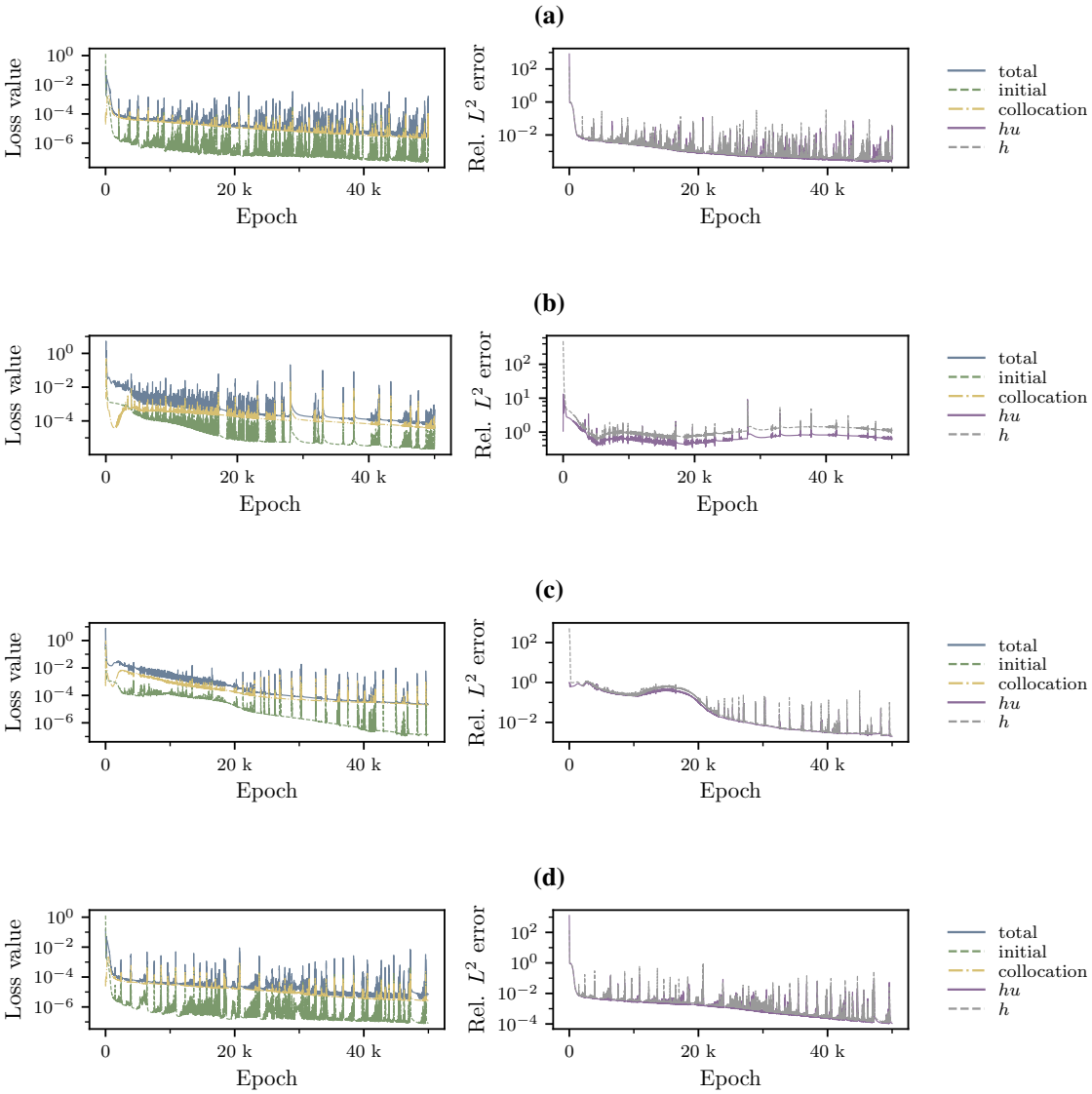

Figure 5.2: Scenario 1 — Evolution of losses and prediction errors of different PINN forms. The left column shows the losses values, and the right column the relative L^2 error between prediction and simulation at 10K randomly sampled locations over the training of 50K epochs. Figure **(a)** illustrates the training of the conservative form f_θ^{C} defined in 4.3. Figure **(b)** represents the scalar-valued form f_θ^{ϕ} defined in 4.7, and Figures **(c)** and **(d)** illustrate the training of the non-conservative form f_θ^{NC}, as defined in 4.8 with bias set to 0.0 and 1.01, respectively.

Method	total err.	h err.	hu err.	mass eq.	mom. eq.	epoch
LRA	2.2×10^{-3}	2.2×10^{-3}	2.1×10^{-3}	2.8×10^{-5}	1.4×10^{-5}	49,729
Siren	8.2×10^{-3}	9.6×10^{-3}	6.9×10^{-3}	1×10^{-5}	6×10^{-6}	48,163
L-LAAF	7×10^{-5}	8.1×10^{-5}	5.8×10^{-5}	5.5×10^{-7}	4.7×10^{-7}	49,997
N-LAAF	6.8×10^{-5}	7.1×10^{-5}	6.6×10^{-5}	2.6×10^{-6}	2.3×10^{-6}	49,737
Attention	2.6×10^{-3}	2.6×10^{-3}	2.6×10^{-3}	6.7×10^{-4}	3.5×10^{-4}	39,825

Table 5.4: Scenario 1 — Performances of different PINNs forms, each row corresponds to the network with lowest total validation error achieved after training 50K epochs.

Examining the results shows that the SIREN approach performed lowest followed by the LRA method. The SIREN method (with LRA) requires more iterations, as the validation error did not seem to have fully converged. The attention mechanism also shows worse performance than the baseline. The adaptive activation methods performed best. In the end, both methods achieved similar results, but one can see that the L-LAAF method converged quicker in the first 20K epochs. The results show, that it takes approximately 12 minutes of training to get a total validation error in the order of 10^{-4}. Figures 5.4 and 5.5 show a detailed comparison of the best prediction, provided by the N-LAAF method, and the result achieved by the simulation. As one can see, no difference is visible between the simulation and prediction results in the presented scale.

5.2 Scenario 2 — Dam Break

The following considers an idealized one-dimensional dam break scenario, which turns out to be more challenging to solve than the one above, as it contains discontinuities in the initial condition and shocks that propagate through space as time elapses. The initial value problem considered here is:

$$h(x,0) = \begin{cases} 3 & x \leq 0 \\ 1 & x > 0 \end{cases}$$
$$hu(x,0) = 0$$

where the domain is restricted to $x \in [-5, 5]$, and the considered time scale is limited to $t \in [0, 1]$. In the beginning, the water surface is at rest. The imaginary dam sits at location $x = 0$ and has infinitesimal width. It breaks immediately and disappears for all $t > 0$. After the dam break, water streams from the higher, former dammed region at $x \leq 0$ into the lower region at $x > 0$, which leads to the formation of a flood wave at the streaming front, while the water height decreases in the previously dammed region. Figure 5.6 illustrates the scenario at three timesteps extracted from a Pyclaw simulation with a discretization of 500 cells.

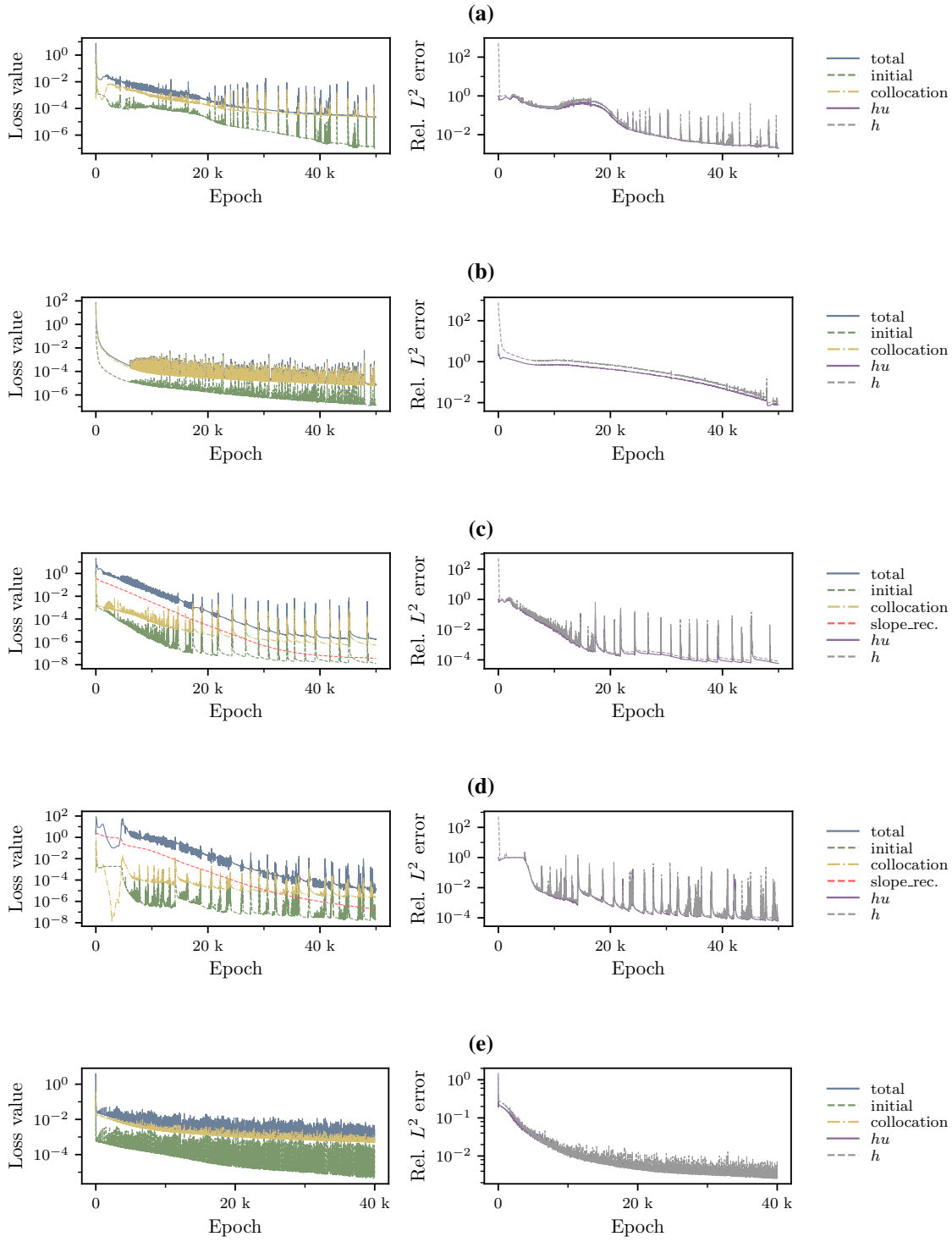

Figure 5.3: Scenario 1 — Comparison of optimization strategies. The left column shows the evolution of loss values, and the right column the relative L^2 error between prediction and simulation at 10K randomly sampled locations. Figure **(a)** illustrates the LRA method. Figure **(b)** represents the SIREN approach, Figures **(c)** and **(d)** illustrate the adaptive activation methods L-LAAF and N-LAAF, and **(e)** shows the training the attention based approach with fixed weights after pretraining of 10K epochs.

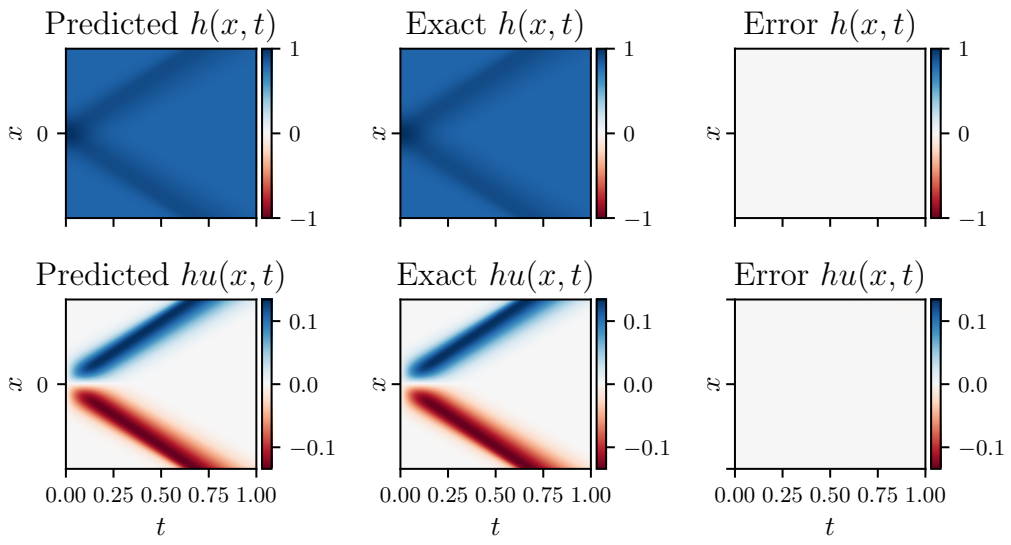

Figure 5.4: Scenario 1 — Comparison between the best prediction achieved with N-LAAF (at epoch 49737 with relative L^2 error of 6.8×10^{-5}) and simulation results (exact). The third column denotes the point-wise difference between prediction and the exact value.

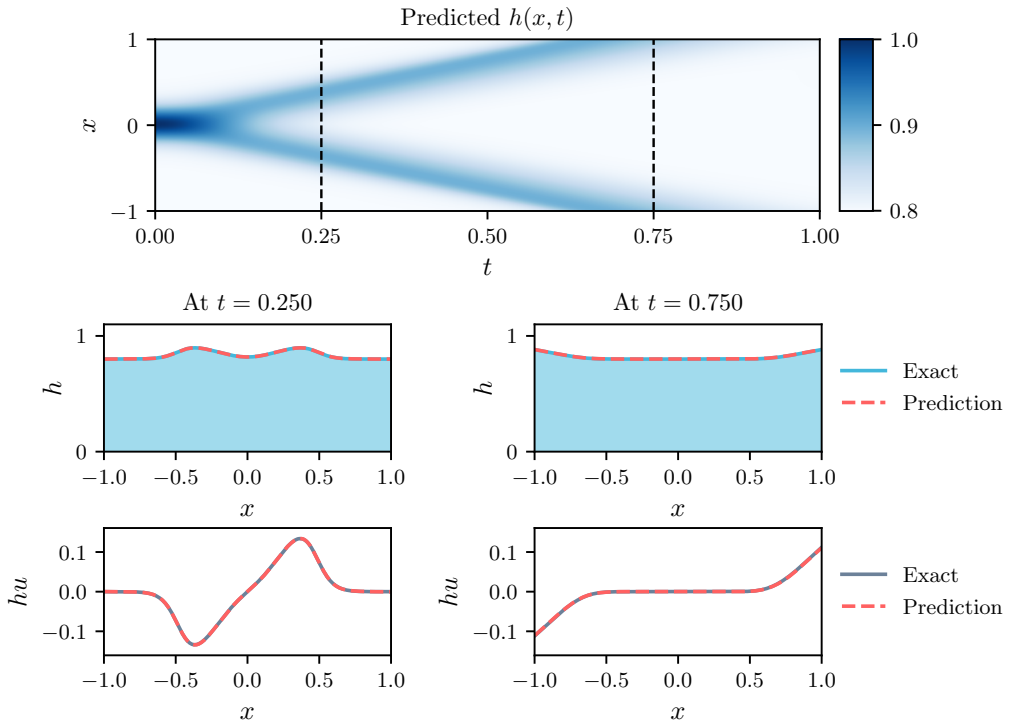

Figure 5.5: Scenario 1 — Comparison between best prediction achieved with N-LAAF (at epoch 49737 with relative L^2 error of 6.8×10^{-5}) and simulation results (exact) at two different time steps. *(Top)* shows the predicted $h(x,t)$ for the whole domain $(x,t) \in [-1,1] \times [0,1]$ along with two marked timesteps. *(Bottom)* illustrates a comparison between the predicted and the simulated height and momentum at those timesteps.

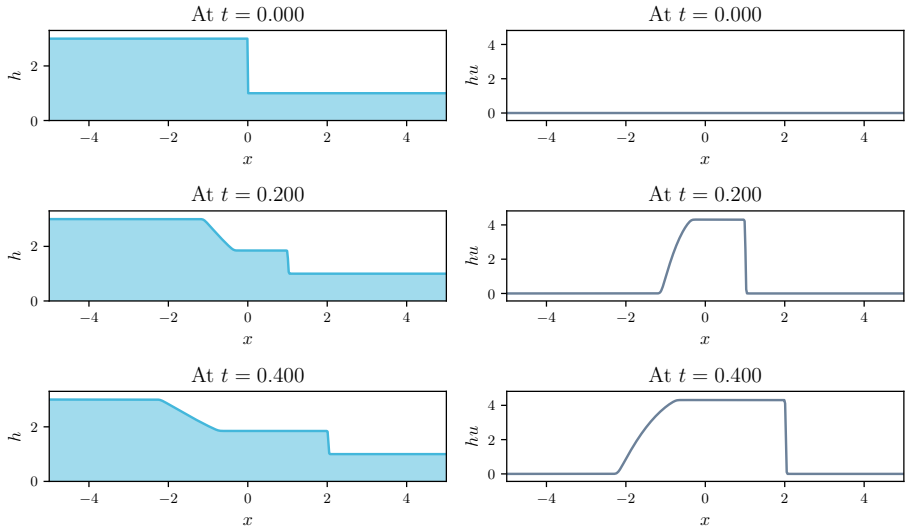

Figure 5.6: Scenario 2 — Three timesteps extracted from a Pyclaw [KMA+12] simulation with a spatial discretization consisting of 500 cells, and gravity g set to 9.8. The left column shows the water depth h and the right column the momentum hu.

5.2.1 PINN Results

Similar to the previous scenario, I considered training different network architectures on the simulation dataset to see which performance can be expected when solving the equations within the PINN framework. It shows that the network with a depth of 5 and width of 20 converges faster than in the previous case, leading to a total validation error of 2.31×10^{-5} for the hyperbolic tangent, and 6.77×10^{-5} for the sine activation function after 4K epochs. However, the solution achieved with the PINN in the non-conservative form after training 50K epochs shows a much higher validation error for all of the five optimization strategies. Table 5.5 summarizes the models with smallest validation error. Again a learning rate of 10^{-3} was chosen. Analyzing the training process, illustrated in Figure 5.7, shows that the PDE loss converges fast at the beginning but then begins oscillating strongly for the hyperbolic tangent based approaches and even diverges in case of the adaptive activation methods. Also, note that the best validation error is reached within the first 10K epochs in all approaches. Typically, strong fluctuations with no progress or even divergence of the loss functions are a sign that the learning rate is set too high. However, I also tried smaller learning rates and trained for more epochs, but oscillations still occurred, and there was no improvement compared to the results presented here. Additionally I tried a step-based learning rate scheduler, however, it needed carefully adjusted step sizes, which is unpractical in general, and the solution could not be improved. When taking a look at Figures 5.8 and 5.9, which show the best model, achieved with the time-adaptive sampling approach described later, it shows that a large portion of the error is located at the steep front of the wave. Therefore, the hope was that the attention-based approach, which focuses the training on regions with higher losses, could improve the results. Although I observed an increase of weighting around the discontinuities as expected, the attention-based method performs again lower than the baseline approach, as Table 5.5 shows.

Method	total err.	h err.	hu err.	mass eq.	mom. eq.	epoch
LRA	1.9×10^{-2}	9.7×10^{-3}	2.8×10^{-2}	2.3×10^{-1}	1.4×10^{-1}	17,524
Siren	3.1×10^{-2}	1.6×10^{-2}	4.7×10^{-2}	8.1×10^{-2}	2×10^{-2}	6,224
L-LAAF	1.7×10^{-2}	6.9×10^{-3}	2.7×10^{-2}	1.5×10^{-1}	3.4×10^{-1}	14,093
N-LAAF	1.8×10^{-2}	7.7×10^{-3}	2.7×10^{-2}	1.1×10^{-1}	5.5×10^{-2}	10,423
Attention	2.3×10^{-2}	1.1×10^{-2}	3.5×10^{-2}	6.4	1.7	13,270

Table 5.5: Scenario 2 — Summary of the best predictions achieved with different optimization strategies. Each row corresponds to the network with lowest total validation error, achieved after training 50K epochs using the respecting optimization method.

Similar to the previous approach, the adaptive approaches performed best, but this time the error is more than two orders of magnitude higher than in the previous case. Looking at the relative L^2 errors of the predicted quantities height h and momentum hu in Figure 5.7, one can see that they differ during training in all cases. Especially for the hyperbolic tangent based approaches a clear gap is visible. That implies that the training is biased and prefers decreasing the height term in this case. Further investigations are necessary to compensate for this effect.

To further improve the results, I learned the L-LAAF based model with the time-adaptive sampling which is illustrated on the left in Figure 4.1. The idea was to spend more time on the region close to initial conditions before considering the whole domain. Therefore I trained another 50K epochs while starting with the temporal domain limited by $t = 0.1$ and increased the time by 0.1 whenever a maximum number of 5000 epochs has passed (until $t = 1$ is reached). The results are presented in Figures 5.8 and 5.9. The achieved validation error was 1.3×10^{-2} in epoch 43442 and thus showed an improvement of 0.4×10^{-2} to the model trained on the whole domain in each iteration.

5.3 Scenario 3 — Bathymetry

This section considers an initial value problem with a similar depth perturbation as in scenario 1 but with stronger gravity and varying bathymetry. The initial conditions of the problem are defined as:

$$
\begin{aligned}
b(x) &= 0.8 \cdot \exp\left(\frac{-x^2}{0.4}\right) - 1.0 \\
h(x,0) &= 0.2 \cdot \exp\left(\frac{-(x+0.4)^2}{0.4}\right) - b(x) \\
u(x,0) &= 0
\end{aligned}
$$

Here, b(x) denotes the bathymetry, which describes an underwater bump. Similar to the first case, the spatial domain is limited to $x \in [-1, 1]$ and the considered timescale is $t \in [0, 1]$. The scenario is an adapted version of an example test case provided by the Pyclaw simulation software [KMA+12]. Figure 5.10 illustrates the progression of height and momentum at three different times. As one can see in the last row, at $t = 0.6$, the wavefront is much steeper than in scenario 1, which results from the presence of the bathymetry and increased gravity.

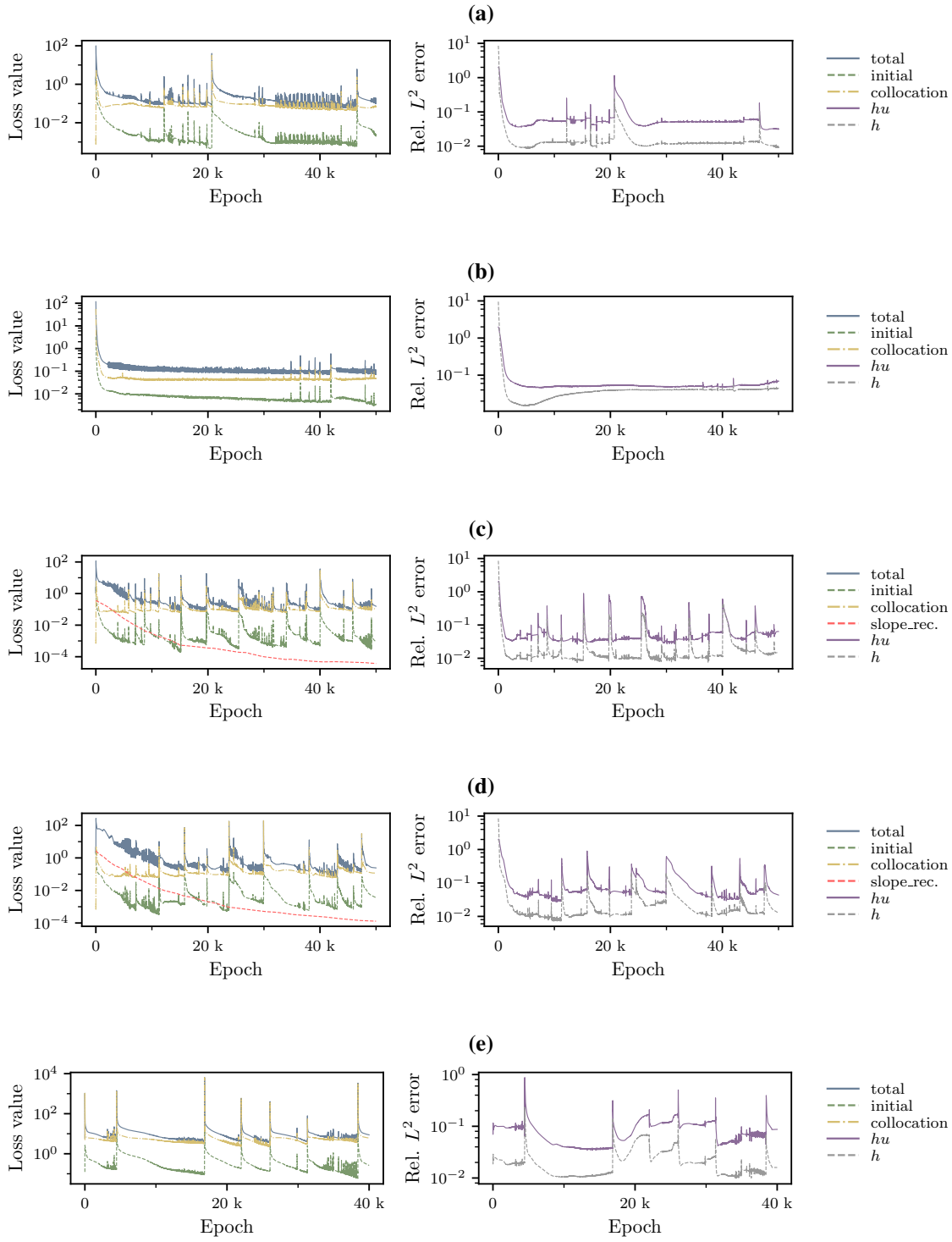

Figure 5.7: Scenario 2 — Comparison of optimization strategies. The left column shows the evolution of loss values, and the right column the relative L^2 error between prediction and simulation at 10K randomly sampled locations. Figure **(a)** illustrates the LRA method. Figure **(b)** represents the SIREN approach, Figures **(c)** and **(d)** illustrate the adaptive activation methods L-LAAF and N-LAAF, and **(e)** shows the training the attention based approach with fixed weights after pretraining of 10K epochs.

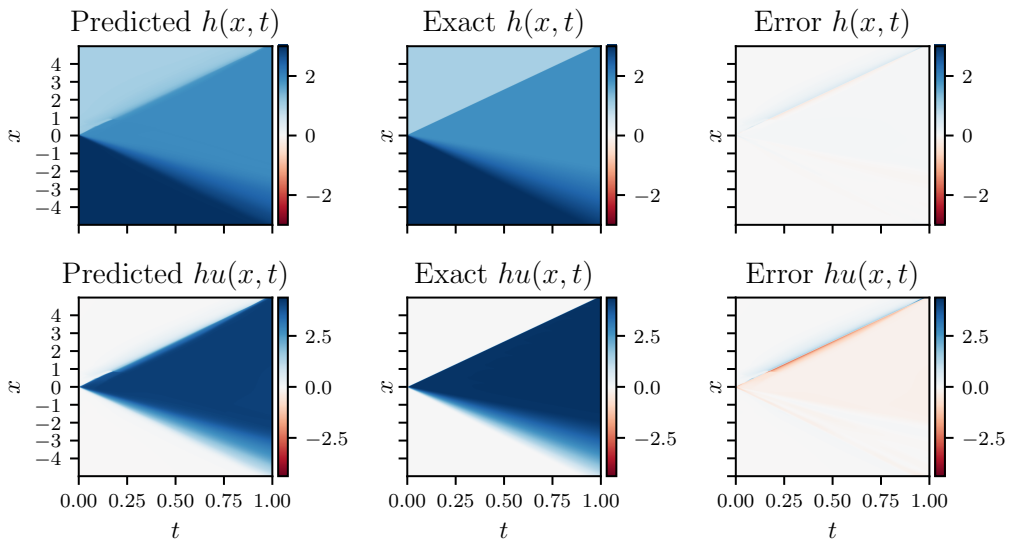

Figure 5.8: Scenario 2 — Comparison between the best prediction achieved with L-LAAF and time-adaptive sampling (at epoch 43442 with relative L^2 error of 1.3×10^{-2}) and simulation results (exact). The third column denotes the point-wise difference between prediction and the exact value.

5.3.1 PINN Results

Like in the previous scenario I trained different network architectures first on the simulation data to get a feel for the expected error when using the PINN approach. This time, it showed that after 4K epochs the network with a width of 20 and depth of 5 and hyperbolic tangent activation function only achieved a validation error close to 10^{-2}. As the depth of the fluid h in this case also must implicitly represent the bathymetry, I decided to increase the approximation capabilities of the network and picked a large model with 50 neurons in each hidden layer and the same depth of 5. This network reached a value of close to 10^{-3} for both the sine and hyperbolic tangent activation function. Again, I trained all five optimization strategies. The results of the best models are summarized in Table 5.6 and the evolution of the loss values and validation metrics are illustrated in Figure 5.11. Considering the validation errors of the best models listed in Table 5.6, it shows that the SIREN based approach performed worst again. Surprisingly, this time the baseline and the attention-based model outperformed the L-LAAF approach. Especially the attentio-based model shows further potential but has been stopped at this point. The evolution of the loss terms shows, that all the mentioned approaches yield a reasonable convergence, and models with errors close to the best ones described in Table 5.6 can be easily achieved without considering any validation data. The results also show, that the N-LAAF based method achieved the best performance in the shortest amount of training cycles. It even produces good results for the depth within the first 10K epochs. However, especially the validation errors begin to oscillate similarly as in the dam break scenario. To compensate for the oscillations one might consider using a smaller learning rate, but this has not been examined. Also, considering the behavior of the PDE and initial loss provide a hint when spikes in the validation error occur and one could stop the training process accordingly, but this is not always a valid metric as, towards the end, the solution diverges while the PDE does not change except for the oscillations. Similar to the dam break problem I tried the time adaptive sampling

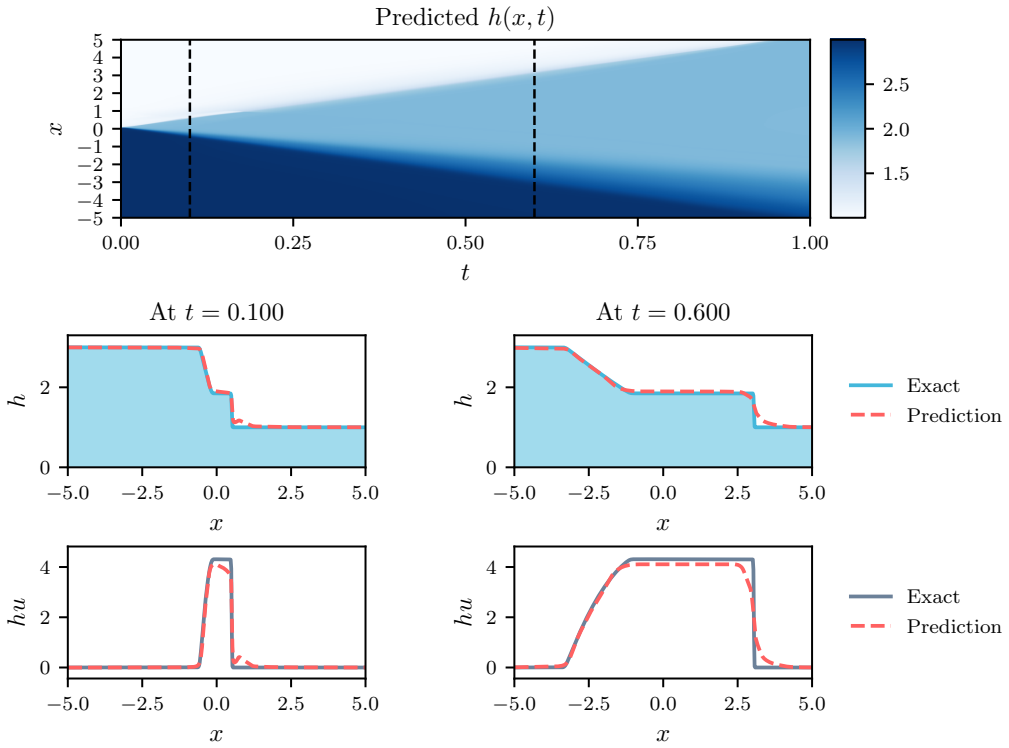

Figure 5.9: Scenario 2 — Comparison between best prediction achieved with L-LAAF and time-adaptive sampling (at epoch 43442 with relative L^2 error of 1.3×10^{-2}) and simulation results (exact) at two different time steps. *(Top)* shows the predicted $h(x,t)$ for the whole domain $(x,t) \in [-1,1] \times [0,1]$ along with two marked timesteps. *(Bottom)* illustrates a comparison between the predicted and the simulated height and momentum at those timesteps.

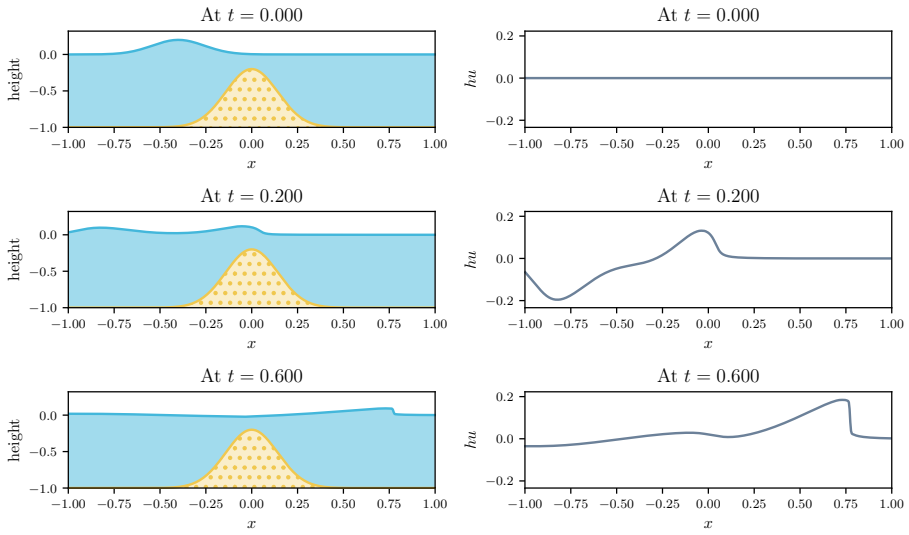

Figure 5.10: Scenario 3 — Three timesteps extracted from a Pyclaw [KMA+12] simulation with a spatial discretization consisting of 500 cells, and gravity g set to 3.5. The left column shows the water depth h and the right column the momentum hu.

Method	total err.	h err.	hu err.	mass eq.	mom. eq.	epoch
LRA	2.4×10^{-2}	1.6×10^{-3}	4.6×10^{-2}	6.3×10^{-4}	3.5×10^{-3}	44,349
Siren	1.1×10^{-1}	3.9×10^{-3}	2.1×10^{-1}	8.3×10^{-4}	2.8×10^{-3}	46,070
L-LAAF	3×10^{-2}	1.6×10^{-3}	5.8×10^{-2}	1×10^{-4}	4×10^{-4}	49,663
N-LAAF	8.9×10^{-3}	4.1×10^{-4}	1.7×10^{-2}	2.6×10^{-4}	4.5×10^{-4}	33,768
Attention	1.8×10^{-2}	1.2×10^{-3}	3.5×10^{-2}	4.6×10^{-3}	4.4×10^{-2}	39,966

Table 5.6: Scenario 3 — Summary of the best predictions achieved with different optimization strategies. Each row corresponds to the network with lowest total validation error, achieved after training 50K epochs using the respecting optimization method.

approach which increases the sampling domain of the collocation points every 5K epochs. However, the results did not improve this time. Like in the dam break problem, one can identify a large gap between the performance of the height and momentum when considering the validation metrics in the right column of Figure 5.11. It is not clear how much this affects the training performance and should be examined further. Also the comparison plots in Figures 5.12 and 5.13 show the different performance of the two quantities. It also shows that the errors are mainly located around edges with increased slopes. Regarding the upper left image in Figure 5.12, one can also observe that the bathymetry was learned well by depth forming the white band. Although the derivative of the bathymetry is well defined in this case, it is worth mentioning, that one may also provide a more realistic bathymetry which is only given at discrete locations by using derivatives from an interpolation method to compute its spatial derivative $\frac{\partial b}{\partial x}$ needed for the PINN.

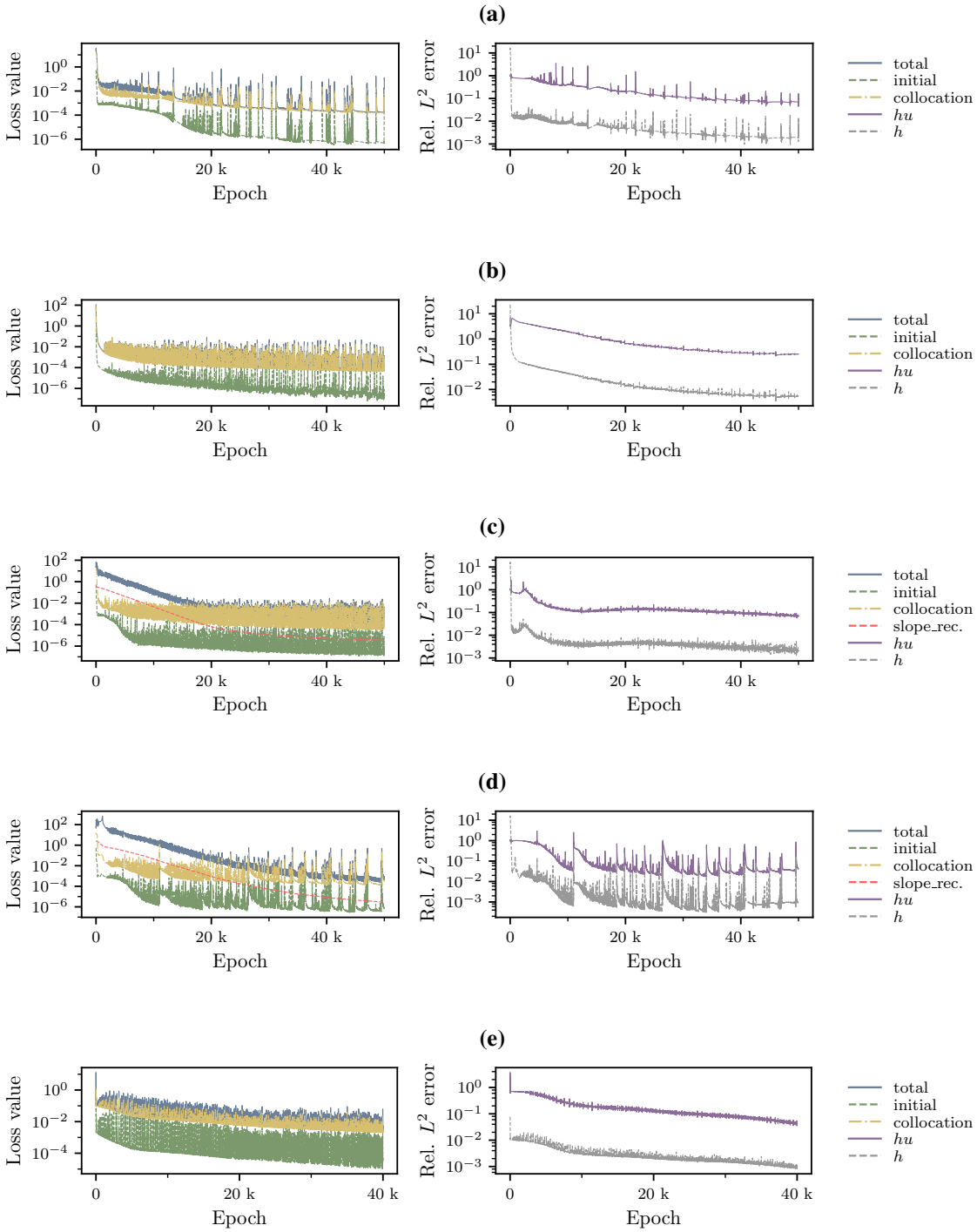

Figure 5.11: Scenario 3 — Comparison of optimization strategies. The left column shows the evolution of loss values, and the right column the relative L^2 error between prediction and simulation at 10K randomly sampled locations. Figure **(a)** illustrates the LRA method. Figure **(b)** represents the SIREN approach, Figures **(c)** and **(d)** illustrate the adaptive activation methods L-LAAF and N-LAAF, and **(e)** shows the training the attention based approach with fixed weights after pretraining of 10K epochs.

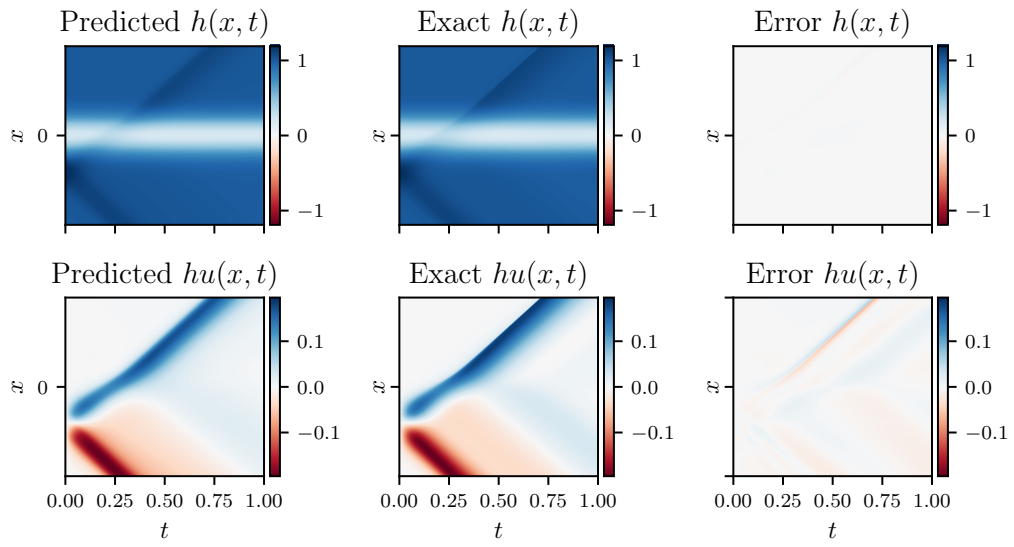

Figure 5.12: Scenario 3 — Comparison between the best prediction achieved with N-LAAF (at epoch 33768 with relative L^2 error of 8.9×10^{-3}) and simulation results (exact). The third column denotes the point-wise difference between prediction and the exact value.

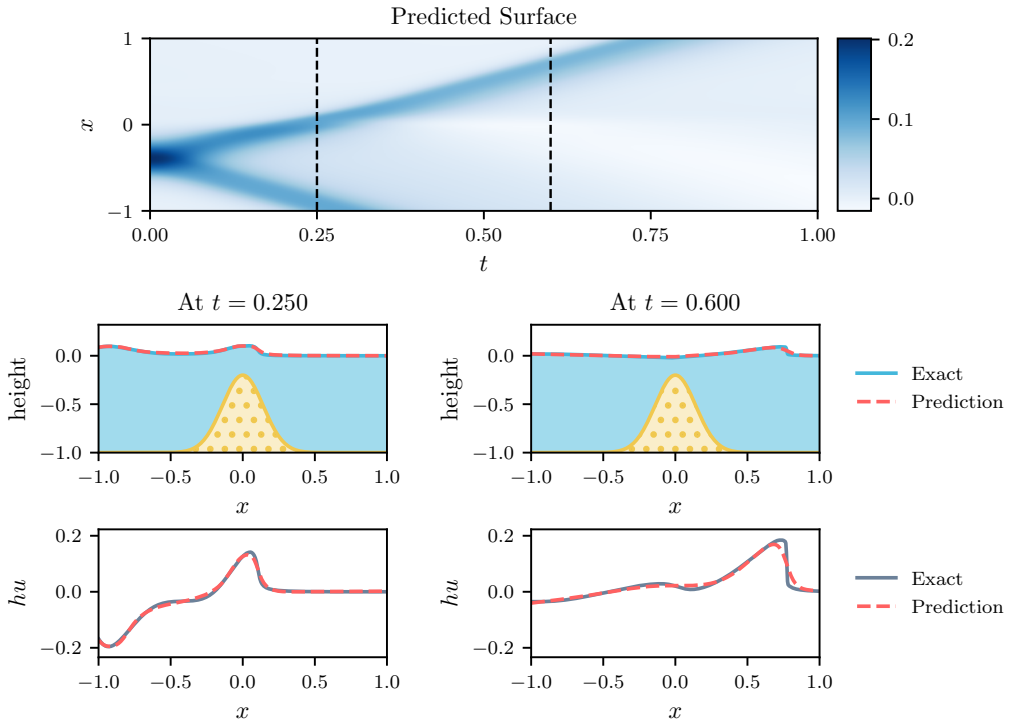

Figure 5.13: Scenario 3 — Comparison between best prediction achieved with N-LAAF (at epoch 33768 with relative L^2 error of 8.9×10^{-3}) and simulation results (exact) at two different time steps. *(Top)* shows the predicted $h(x,t)$ for the whole domain $(x,t) \in [-1,1] \times [0,1]$ along with two marked timesteps. *(Bottom)* illustrates a comparison between the predicted and the simulated height and momentum at those timesteps.

6 Conclusion and Outlook

This thesis investigated the application of the PINN framework to solve free-surface flow problems modeled by the one dimensional SWE. The results are presented at three test cases: Two over a flat bathymetry, an initial depth perturbation with moderate steepening of wavefronts, and an idealized dam break scenario, and an initial depth perturbation with varying bathymetry and stronger steepening wavefronts. For each of the three cases, different network architectures were trained to approximate data points extracted from a high-resolution numerical solution to get an impression of how many neurons and layers are needed to achieve a satisfying result. It showed, that a network of width 20 and depth 5 is sufficient to achieve reasonable results for the first and second scenario. As expected, varying bathymetry increases the complexity, and more parameters are necessary to achieve similar results. The accurate location of the free surface is given by the sum of bathymetry and the fluid depth. Therefore the bathymetry must be implicitly learned by the model's depth prediction. Here, a network with a width of 50 and depth of 5 showed similar results as in the first two cases.

The SWE allow different formulation of the PINN. I presented three different versions, including a formulation that implicitly fulfills mass conservation and, thus, eliminates the corresponding penalization term in the PDE loss. The results show that the non-conservative formulation is best suited and can be applied most flexibly to different initialization schemes, where the other might fail training due to high values or division by zero errors. However, more research is needed to fully exploit the formulation that implicitly fulfills the mass conservation equation.

For each of the three previously described scenarios, five state-of-the-art optimization techniques and a time-adaptive sampling strategy were explored to improve results and speed up convergence. It showed that for the first scenario with moderate steepening wavefronts, the locally adaptive activation methods presented in [JKK19] drastically speed up the convergence process and yielded a relative L^2 error in the order of 10^{-4} after 10K iterations of full-batch training with the ADAM optimizer. However, the other two scenarios could not reach a similar performance. Also, the performance difference between the five methods was not as large as in the first scenario. For the dam break scenario, the best result has a relative L^2 error of 1.3×10^{-2} using the L-LAAF method with time-adaptive sampling. The best model for the third scenario achieved an error of 8.9×10^{-3} employing the N-LAAF method. However, the attention-based method showed some prospects which could further improve the result when run longer than 50K epochs, which has not been tested. In both cases, the largest error emerged at steep wavefronts, with larger gradients and high-frequency components. Although some of the optimization methods are specifically tailored to reduce the error in such regions, especially in the dam break scenario, they did not show much improvement compared to the baseline approach, which uses the LRA method. Therefore, more research is needed to further improve the results in cases with strong steepening wavefronts.

Outlook

Throughout the work, unresolved challenges, ideas for improvements, and prospects emerged, which provide a good starting point for future work. Beginning with the open challenges, the PINN in scalar-valued form showed that a model for the SWE can be constructed that always fulfills the mass-conservation equation and removes its penalization term in the PINN. Although the error of both conservation equations become small in all cases, except for the dam break scenario, the model is more physically sound if the equation is always fulfilled. However, to prevent divisions by zero, especially at the beginning of the training process, a tailored initialization scheme is necessary, which requires further investigation. In general, an inspection of the initialization scheme could further speed up the training process, i.e. a meta-learning approach could be applied to yield better initialization of the weights and biases. Another open challenge lies in the different training speeds of the two quantities depth h and momentum hu, which can be observed in the dam break and bathymetry scenario. It seemed that the network is biased in direction of the depth and further investigation is needed to fully understand the influence on the training process.

As the PINN framework is a relatively new approach that is not fully understood and is under strong development, many improvements to the learned models were just presented recently. Especially two methods of Jagtap et al. stood out in the context of the SWE, namely, the conservative-PINN approach [JKK20b] and their newest development eXtended PINNs [JK20]. Both methods allow decomposing the solution domain which enables distributed learning similar to well-known domain decomposition methods for classical numerical models. They show improvements to the learned models, where the former specializes to PDEs in conservative form, the latter extends to the general case and allows arbitrary decomposition of the domain in space and time.

A preceding step could be an extension of the proposed methods to 2D problems. Other approaches successfully trained the 2D hyperbolic conservative wave equation, however, it shows that a large amount of computational effort is necessary i.e. 24 hours on a modern high-performance graphics card with 24 GB of memory, although time-adaptive sampling was employed [SMB+20]. Also, to enable more realistic scenarios, boundary conditions such as inflow, outflow, and reflective could be employed, along with wetting and drying. Especially the reflective boundary conditions that emerge when static obstacles prevent a continuous flow are challenging as they typically rely on reflecting values at the cell boundary in a finite volume scheme. However, applying this behavior to a cell-free approach is not straight forward.

Bibliography

[ABC+16] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, X. Zheng. "TensorFlow: A system for large-scale machine learning". In: *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2016*. 2016. ISBN: 9781931971331. arXiv: 1605.08695 (cit. on pp. 24, 28).

[BC96] S. W. Bova, G. F. Carey. "An entropy variable formulation and applications for the two-dimensional shallow water equations". In: *International Journal for Numerical Methods in Fluids* (1996). ISSN: 02712091. DOI: 10.1002/(sici)1097-0363(19960715)23:1<29::aid-fld411>3.3.co;2-l (cit. on p. 21).

[Bel66] R. Bellman. "Dynamic programming". In: *Science* 153.3731 (1966), pp. 34–37 (cit. on p. 30).

[Cla] Clawpack Development Team. *Shallow water Riemann solvers in Clawpack*. URL: https://www.clawpack.org/riemann/Shallow%7B%5C_%7Dwater%7B%5C_%7DRiemann%7B%5C_%7Dsolvers.html (cit. on p. 22).

[Cla20] Clawpack Development Team. *Clawpack software*. 2020. DOI: https://doi.org/10.5281/zenodo.4025432. URL: http://www.clawpack.org (cit. on pp. 18, 22).

[Cli] Clint Dawson and Christopher M. Mirabito. *The Shallow Water Equations*. URL: https://users.oden.utexas.edu/%7B~%7Darbogast/cam397/dawson%7B%5C_%7Dv2.pdf (cit. on p. 18).

[DHS11] J. Duchi, E. Hazan, Y. Singer. "Adaptive subgradient methods for online learning and stochastic optimization." In: *Journal of machine learning research* 12.7 (2011) (cit. on p. 27).

[DLK+13] O. Delestre, C. Lucas, P. A. Ksinant, F. Darboux, C. Laguerre, T. N. Vo, F. James, S. Cordier. "SWASHES: A compilation of shallow water analytic solutions for hydraulic and environmental studies". In: *International Journal for Numerical Methods in Fluids* (2013). ISSN: 02712091. DOI: 10.1002/fld.3741. arXiv: 1110.0288 (cit. on pp. 21, 22).

[DS19] Y. N. Dauphin, S. S. Schoenholz. "MetaInit: Initializing learning by learning to initialize". In: *Advances in Neural Information Processing Systems*. 2019 (cit. on p. 45).

[GB10] X. Glorot, Y. Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Journal of Machine Learning Research*. 2010 (cit. on pp. 32, 34, 35).

[Ger05] H. Gerritsen. “What happened in 1953? The Big Flood in the Netherlands in retrospect”. In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 363.1831 (2005), pp. 1271–1291 (cit. on p. 19).

[Hin] Hinton, Geoffrey. *Lecture 6e rmsprop: Divide the gradient by a running average of its recent magnitude*. URL: http://www.cs.toronto.edu/%7B~%7Dtijmen/csc321/slides/lecture%7B%5C_%7Dslides%7B%5C_%7Dlec6.pdf (cit. on pp. 27, 28).

[HS97] S. Hochreiter, J. Schmidhuber. “Long Short-Term Memory”. In: *Neural Computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 24).

[Ima99] R. Iman. “Latin Hypercube Sampling”. In: (1999). DOI: 10.1002/9780470061596.risk0299 (cit. on p. 32).

[Jak06] J. Jakeman. “ON NUMERICAL SOLUTIONS OF THE SHALLOW WATER WAVE EQUATIONS”. Australian National University, 2006. URL: http://docplayer.net/130769044-On-numerical-solutions-of-the-shallow-water-wave-equations.html (cit. on pp. 20, 21).

[JEP+20] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, K. Tunyasuvunakool, O. Ronneberger, R. Bates, A. Žídek, A. Bridgland, C. Meyer, S. A. A. Kohl, A. Potapenko, A. J. Ballard, A. Cowie, B. Romera-Paredes, S. Nikolov, R. Jain, J. Adler, T. Back, S. Petersen, D. Reiman, M. Steinegger, M. Pacholska, D. Silver, O. Vinyals, A. W. Senior, K. Kavukcuoglu, P. Kohli, D. Hassabis. “High Accuracy Protein Structure Prediction Using Deep Learning”. In: *Critical Assessment of Techniques for Protein Structure Prediction (Abstract Book)*. 14. Protein Structure Prediction Center, 2020, pp. 22–24 (cit. on p. 14).

[JK20] A. D. Jagtap, G. E. Karniadakis. “Extended physics-informed neural networks (XPINNs): A generalized space-time domain decomposition based deep learning framework for nonlinear partial differential equations”. In: *Communications in Computational Physics* (2020). ISSN: 19917120. DOI: 10.4208/CICP.OA-2020-0164 (cit. on p. 61).

[JKK19] A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis. “Locally adaptive activation functions with slope recovery term for deep and physics-informed neural networks”. In: *arXiv* (2019). ISSN: 1364-5021. DOI: 10.1098/rspa.2020.0334. arXiv: 1909.12228 (cit. on pp. 15, 37, 60).

[JKK20a] A. D. Jagtap, K. Kawaguchi, G. E. Karniadakis. “Adaptive activation functions accelerate convergence in deep and physics-informed neural networks”. In: *Journal of Computational Physics* 404 (2020). ISSN: 10902716. DOI: 10.1016/j.jcp.2019.109136. arXiv: 1906.01170 (cit. on p. 15).

[JKK20b] A. D. Jagtap, E. Kharazmi, G. E. Karniadakis. “Conservative physics-informed neural networks on discrete domains for conservation laws: Applications to forward and inverse problems”. In: *Computer Methods in Applied Mechanics and Engineering* (2020). ISSN: 00457825. DOI: 10.1016/j.cma.2020.113028 (cit. on p. 61).

[KAT+19] B. Kim, V. C. Azevedo, N. Thuerey, T. Kim, M. Gross, B. Solenthaler. “Deep Fluids: A Generative Network for Parameterized Fluid Simulations”. In: *Computer Graphics Forum* 38.2 (2019), pp. 59–70. ISSN: 14678659. DOI: 10.1111/cgf.13619. arXiv: 1806.02071 (cit. on p. 34).

[Kat19] N. D. Katopodes. *Free-Surface Flow: Shallow-Water Dynamics*. Elsevier, 2019, p. 848. ISBN: 9780128154878. DOI: 10.1016/c2016-0-05220-8. URL: https://linkinghub.elsevier.com/retrieve/pii/C20160052208 (cit. on pp. 18, 20).

[KB17] D. P. Kingma, J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG] (cit. on pp. 27, 28, 36).

[KB20] A. Kratsios, E. Bilokopytov. *Non-Euclidean Universal Approximation*. 2020. arXiv: 2006.02341 [cs.LG] (cit. on p. 25).

[KL19] P. Kidger, T. J. Lyons. "Universal Approximation with Deep Narrow Networks". In: *CoRR* abs/1905.0 (2019). arXiv: 1905.08539. URL: http://arxiv.org/abs/1905.08539 (cit. on p. 25).

[KMA+12] D. I. Ketcheson, K. T. Mandli, A. J. Ahmadia, A. Alghamdi, M. Quezada de Luna, M. Parsani, M. G. Knepley, M. Emmett. "PyClaw: Accessible, Extensible, Scalable Tools for Wave Propagation Problems". In: *SIAM Journal on Scientific Computing* 34.4 (Nov. 2012), pp. C210–C231 (cit. on pp. 21, 22, 42, 43, 50, 51, 54).

[KSH17] A. Krizhevsky, I. Sutskever, G. E. Hinton. "ImageNet classification with deep convolutional neural networks". In: *Communications of the ACM*. Ed. by F. Pereira, C. J. C. Burges, L. Bottou, K. Q. Weinberger. Vol. 60. 6. Curran Associates, Inc., 2017, pp. 84–90. DOI: 10.1145/3065386. URL: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf (cit. on p. 24).

[LeV02] R. J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. 2002. DOI: 10.1017/cbo9780511791253 (cit. on p. 22).

[LG08] R. J. LeVeque, D. L. George. *High-Resolution Finite Volume Methods for the Shallow Water Equations With Bathymetry and Dry States*. Vol. M. September 2008. 2008, pp. 43–73. ISBN: 9789812790910. DOI: 10.1142/9789812790910_0002 (cit. on p. 22).

[LLZC20] W. Li, K. Liu, L. Zhang, F. Cheng. "Object detection based on an adaptive attention mechanism". In: *Scientific Reports* 10 (2020). DOI: 10.1038/s41598-020-67529-x (cit. on p. 39).

[LN89] D. C. Liu, J. Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical Programming* (1989). ISSN: 00255610. DOI: 10.1007/BF01589116 (cit. on p. 40).

[LPF+13] F. Lavigne, R. Paris, L. Frédéric, G. J.-C, J. Morin. "The 2004 Indien Ocean Tsunami". In: 2013, 1135 p. ISBN: 978-94-007-0263-9 (cit. on p. 18).

[Mat] A. Mattas. *mattflow*. URL: https://pypi.org/project/mattflow/ (cit. on p. 22).

[MB20] L. McClenny, U. Braga-Neto. "Self-Adaptive Physics-Informed Neural Networks using a Soft Attention Mechanism". In: (2020). arXiv: 2009.04544. URL: http://arxiv.org/abs/2009.04544 (cit. on pp. 15, 38).

[MS10] P. A. Madsen, H. A. Schäffer. "Analytical solutions for tsunami runup on a plane beach: Single waves, N-waves and transient waves". In: *Journal of Fluid Mechanics* (2010). ISSN: 00221120. DOI: 10.1017/S0022112009992485 (cit. on p. 18).

[Owh15] H. Owhadi. "Bayesian numerical homogenization". In: *Multiscale Modeling and Simulation* (2015). ISSN: 15403467. DOI: 10.1137/140974596. arXiv: 1406.6668 (cit. on p. 29).

[Pen12] S. H. Peng. "1D and 2D numerical modeling for solving dam-break flow problems using finite volume method". In: *Journal of Applied Mathematics* (2012). ISSN: 1110757X. DOI: 10.1155/2012/489269 (cit. on p. 19).

[PGM+19] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, S. Chintala. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf (cit. on pp. 24, 28).

[Ras04] C. E. Rasmussen. "Gaussian Processes in machine learning". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* (2004). ISSN: 16113349. DOI: 10.1007/978-3-540-28650-9_4 (cit. on p. 29).

[RB92] M. Riedmiller, H. Braun. "Rprop-a fast adaptive learning algorithm". In: *Proc. of ISCIS VII), Universitat*. Citeseer. 1992 (cit. on p. 28).

[RC01] E. Rigby, R. Consulting. "ANUGA – A New Free & Open Source Hydrodynamic Model". In: 1992 (2001) (cit. on p. 22).

[RD08] E. Rigby, R. van Drie. "ANUGA: A New Free and Open Source Hydrodynamic Model". In: *the 4th International Conference on Water Resources and Environment Research (ICWRER)* (2008) (cit. on p. 18).

[RHW86] D. E. Rumelhart, G. E. Hinton, R. J. Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536 (cit. on pp. 27, 28).

[RK18] M. Raissi, G. E. Karniadakis. "Hidden physics models: Machine learning of nonlinear partial differential equations". In: *Journal of Computational Physics* (2018). ISSN: 10902716. DOI: 10.1016/j.jcp.2017.11.039. arXiv: 1708.00588 (cit. on p. 29).

[Ros58] F. Rosenblatt. "The perceptron: A probabilistic model for information storage and organization in the brain". In: *Psychological Review* 65.6 (1958), pp. 386–408. ISSN: 0033295X. DOI: 10.1037/h0042519 (cit. on p. 24).

[RPK17a] M. Raissi, P. Perdikaris, G. E. Karniadakis. "Inferring solutions of differential equations using noisy multi-fidelity data". In: *Journal of Computational Physics* (2017). ISSN: 10902716. DOI: 10.1016/j.jcp.2017.01.060. arXiv: 1607.04805 (cit. on p. 29).

[RPK17b] M. Raissi, P. Perdikaris, G. E. Karniadakis. "Machine learning of linear differential equations using Gaussian processes". In: *Journal of Computational Physics* (2017). ISSN: 10902716. DOI: 10.1016/j.jcp.2017.07.050. arXiv: 1701.02440 (cit. on p. 29).

[RPK18] M. Raissi, P. Perdikaris, G. E. Karniadakis. "Numerical Gaussian processes for time-dependent and nonlinear partial differential equations". In: *SIAM Journal on Scientific Computing* (2018). ISSN: 10957197. DOI: 10.1137/17M1120762. arXiv: 1703.10230 (cit. on p. 29).

[RPK19] M. Raissi, P. Perdikaris, G. E. Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707. ISSN: 10902716. DOI: 10.1016/j.jcp.2018.10.045. URL: https://doi.org/10.1016/j.jcp.2018.10.045 (cit. on pp. 14, 15, 29, 34, 38).

[Rud16] S. Ruder. "An overview of gradient descent optimization algorithms". In: *CoRR* abs/1609.0 (2016). arXiv: 1609.04747. URL: http://arxiv.org/abs/1609.04747 (cit. on p. 27).

[RYK18] M. Raissi, A. Yazdani, G. E. Karniadakis. "Hidden fluid mechanics: A navier-stokes informed deep learning framework for assimilating flow visualization data". In: *arXiv* (2018). arXiv: 1808.04327 (cit. on p. 14).

[RYK20] M. Raissi, A. Yazdani, G. E. Karniadakis. "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations". In: *Science* 367.6481 (2020), pp. 1026–1030. ISSN: 10959203. DOI: 10.1126/science.aaw4741 (cit. on p. 32).

[SK11] O. San, K. Kara. "High-order acurate spectral difference method for shallow water equations". In: *International Journal of Research and Reviews in Applied Sciences* 6 (2011) (cit. on p. 21).

[SLA12] J. Snoek, H. Larochelle, R. P. Adams. "Practical Bayesian optimization of machine learning algorithms". In: *Advances in Neural Information Processing Systems*. 2012. ISBN: 9781627480031. arXiv: 1206.2944 (cit. on p. 43).

[SMB+20] V. Sitzmann, J. N. Martel, A. W. Bergman, D. B. Lindell, G. Wetzstein, S. University. "Implicit Neural Representations with Periodic Activation Functions". In: *arXiv* (2020). arXiv: 2006.09661 (cit. on pp. 15, 37, 61).

[SMBM19] J. Schmidt, M. R. Marques, S. Botti, M. A. Marques. "Recent advances and applications of machine learning in solid-state materials science". In: *npj Computational Materials* 5.1 (2019). ISSN: 20573960. DOI: 10.1038/s41524-019-0221-0. URL: http://dx.doi.org/10.1038/s41524-019-0221-0 (cit. on p. 14).

[TT09] E. F. Toro, E. F. Toro. "The HLL and HLLC Riemann Solvers". In: *Riemann Solvers and Numerical Methods for Fluid Dynamics*. 2009. DOI: 10.1007/b79761_10 (cit. on p. 22).

[VSP+17] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, I. Polosukhin. "Attention is all you need". In: *Advances in Neural Information Processing Systems*. 2017. arXiv: 1706.03762 (cit. on p. 39).

[Wer75] P. J. Werbos. *Beyond Regression: New Tools for Prediction and Analysis in the Behavioral Sciences*. Harvard University, 1975. URL: https://books.google.de/books?id=z81XmgEACAAJ (cit. on pp. 24, 28).

[WLB+92] J. J. Westerink, R. A. Luettich, A. M. Baptists, N. W. Scheffner, P. Farrar. "Tide and Storm Surge Predictions Using Finite Element Model". In: *Journal of Hydraulic Engineering* (1992). ISSN: 0733-9429. DOI: 10.1061/(asce)0733-9429(1992)118:10(1373) (cit. on p. 19).

[WTP20] S. Wang, Y. Teng, P. Perdikaris. "Understanding and Mitigating Gradient Pathologies in Physics-Informed Neural Networks". In: *arXiv* (2020), pp. 1–28. arXiv: 2001.04536 (cit. on pp. 15, 30, 36).

[WWW20] H. Wessels, C. Weißenfels, P. Wriggers. “The neural particle method – An updated Lagrangian physics informed neural network for computational fluid dynamics”. In: *Computer Methods in Applied Mechanics and Engineering* 368 (2020), pp. 1–21. ISSN: 00457825. DOI: 10.1016/j.cma.2020.113127. arXiv: 2003.10208 (cit. on pp. 14, 15).

[WYP20] S. Wang, X. Yu, P. Perdikaris. “When and why Pinns Fail to Train: A Neural Tangent Kernel Perspective”. In: *arXiv* (2020), pp. 1–29. arXiv: 2007.14527 (cit. on pp. 15, 30).

[WZ20] C. L. Wight, J. Zhao. “Solving Allen-Cahn and Cahn-Hilliard Equations using the Adaptive Physics Informed Neural Networks”. In: *arXiv* (2020), pp. 1–25. arXiv: 2007.04542 (cit. on pp. 14, 15, 30, 32).

[ZR03] C. Zoppou, S. Roberts. “Explicit Schemes for Dam-Break Simulations”. In: *Journal of Hydraulic Engineering* (2003). ISSN: 0733-9429. DOI: 10.1061/(asce)0733-9429(2003)129:1(11) (cit. on p. 21).

[ZZX+20] R. Zhang, R. Zen, J. Xing, D. M. S. Arsa, A. Saha, S. Bressan. “Hydrological Process Surrogate Modelling and Simulation with Neural Networks”. In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. 2020. ISBN: 9783030474355. DOI: 10.1007/978-3-030-47436-2_34 (cit. on p. 15).

All links were last followed on January 22, 2021.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature