Institute of Parallel and Distributed Systems

University of Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Bachelorarbeit

# Machine learning of fluid-structure interaction

Axel Schumacher

| | |
|---|---|
| **Course of Study:** | Simulation Technology |
| **Examiner:** | Prof. Dr. rer. nat. habil. Miriam Schulte |
| **Supervisor:** | Amin Totounferoush, M.Sc. |
| **Commenced:** | June 22, 2020 |
| **Completed:** | July 24, 2021 |

## Abstract

For the simulation of fluid-structure interaction, mostly computational intensive methods are used until today, which achieve a very high accuracy, but are usually very expensive. The data-based approach presented here uses training data from the first few time steps of a classical simulation to train neural networks that provide a good prediction for subsequent time steps. The multiphysics domain of fluid flow and structural deformation is split into two single-physics domains. Each solver uses the solutions of the last time steps to predict the solution in the next time step in its own domain. The coupled solvers exchange data via a coupling library, which also provides the control of the simulation.

This work aims to show, on the one hand, that the extension of machine learning for solving coupled partial differential equations of fluid-structure interaction is an endeavor that can scientifically tie in with the success of artificial intelligence in many other fields and, on the other hand, to show challenges that this extension presents. These challenges are important for future work on this topic.

Mainly, deep neural networks are implemented, which consist of a combination of convolutional neural network, recurrent neural network and fully connected layers. This can be used to learn both spatial and temporal dependencies in the data. Based on the simulation of a fluid flow through a one-dimensional elastic tube, it is shown that the predicted solutions agree to a large extent with the classical numerical solution. Special attention was paid to the fact that the presented solvers can be easily transferred to other problems and are suitable to be combined with classical numerical methods, in order to accelerate the coupling convergence of these methods by predicting an accurate initial guess.

## Zusammenfassung

Für die Simulation von Fluid-Struktur-Interaktion werden bis heute zumeist sehr rechenintensive Verfahren angewendet, welche eine sehr hohe Genauigkeit erreichen, dafür aber meist sehr aufwändig sind. Der hier vorgestellte datenbasierte Ansatz nutzt Trainingsdaten aus den wenigen ersten Zeischritten einer klassischen Simulation um neuronale Netzwerke zu trainieren, welche eine gute Vorhersage für folgende Zeitschritte liefern. Das Multiphysik-Gebiet aus Fluidströmung und Strukturdeformation wird aufgeteilt in zwei Einzelphysik-Gebiete. Jeder Löser verwendet die Lösungen der letzten Zeitschritte um damit die Lösung im nächsten Zeitschritt in seinem eigenen Gebiet vorherzusagen. Die gekoppelten Löser tauschen Daten dabei über eine Kopplungsbibliothek aus, welche auch die Steuerung der Simulation übernimmt.

Diese Arbeit soll einerseits zeigen, dass die Erweiterung von maschinellem Lernen auf das Lösen gekoppelter, partieller Differentialgleichungen der Fluid-Struktur-Interaktion ein Vorhaben ist, welches wissenschaftlich an den Erfolg von künstlicher Intelligenz in vielen anderen Bereichen anknüpfen kann und andererseits Herausforderungen zeigen, die diese Erweiterung darstellt und welche für zukünftige Arbeiten an diesem Thema wichtig sind.

Hauptsächlich werden tiefe neuronale Netzwerke implementiert, welche aus einer Kombination aus faltendem neuronalem Netzwerk, rekurrentem neuronalen Netzwerk und vollständig verbundenem neuronalem Netzwerk bestehen. Damit können sowohl räumliche als auch zeitliche Abhängigkeiten in den Daten gelernt werden. Anhand der Simulation eines Fluidstroms durch einen eindimensionalen, elastischen Schlauch wird gezeigt. dass die vorhergesagten Lösungen in weiten Teilen mit der klassischen numerischen Lösung übereinstimmen. Dabei wurde besonders darauf geachtet, dass die vorgestellten Löser einfach auf andere Probleme übertragen werden können und geeignet sind, mit klassischen numerischen Verfahren kombiniert zu werden, um durch die Vorhersage einer genauen Startschätzung die Kopplungskonvergenz dieser Verfahren zu beschleunigen.

# Contents

# List of Figures

# List of Algorithms

# Part I: Propaedeuticum

Machine learning methods have shown promising performance in various scientific fields, from computer vision and product recommendations to disease diagnosis and classification. Expectations are correspondingly high that machine learning methods can also show they have power in areas that are still unexplored with them. One such area where there is still much potential for improved methods is the analysis and finding of meaningful paradigms in data obtained from physical experiments such as the study of fluid-structure interaction. In contrast to classical mathematical modeling and the solution of often partial and nonlinear differential equations derived in the process, a data-based modeling approach has the potential advantage of being able to describe the behavior of a system without such equations.
In this thesis, machine learning approaches will be investigated and their applicability with available methods will be explored. More precisely, fluid-structure interaction problems will be solved by machine learning approaches. The goal is to implement a partitioned approach, where the overall problem is divided into two subdomains according to the physics involved subproblem for the fluid domain and another subproblem for the structural domain. For each of these subproblems, a neural network is trained to replace the classical equation solvers. To solve the coupled fluid-structure interaction problem, the fluid solver and the structure solver are coupled using the coupling library preCICE [BLG+16].

For this purpose, the necessary foundations are established in this first part, and in the second part they are concretely implemented in order to model a selected test case of fluid-structure interaction by machine learning, more precisely with neural networks. Chapter 1 first gives a concise overview of related scientific publications. Chapter 2 shows the background of fluid-structure interaction and its solution methods, chapter 3 introduces the neural networks used in this thesis and their properties, and in chapter 4 possibilities of solving partial differential equations by machine learning are discussed. Thus, the goal of the practical part of this thesis can be defined in chapter 5 and implemented in the second part. Chapter 6 first develops neural networks that predict the solutions for single-physics domains, and Chapter 7 explains how neural networks can be coupled with preCICE. In chapter 8, the preceding parts are merged to solve a coupled fluid-structure interaction problem. Finally, chapter 9 gives an outlook on further possibilities for deeper research in this area and chapter 10 finally resumes this thesis.

# 1 Literature review

Physics-informed neural networks are introduced in [RPK19] and use knowledge gained in advance to improve machine learning techniques by incorporating physical laws or other knowledge about a system to be described into the training of neural networks. This can reduce the space of possible solutions by excluding unphysical solutions and, for example, enforcing conservation of mass in incompressible fluid flows. Even if only a few training samples are available, this can lead to better generalizations. In one example, a physics-informed neural network is presented there, which is supposed to compute parameters of the Navier-Stokes equations in a fluid flow. The mass conservation of the fluid is used as a boundary condition in the modeling. The approach is to use a data set created by numerical solution, from which random samples $(x, y, t)$ are drawn, with which the neural network is trained. Subsequently, the parameters as well as the pressure in the entire flow field at an arbitrary time $t$ are to be predicted. In this sense, continuous-time problems can be solved relatively easily throughout the domain by auxiliary measurements and using the physics behind them. These parameters can even be identified on noisy input data with very small errors. However, it is always clear that machine learning methods should not be seen as a complete replacement for classical numerical methods for solving partial differential equations. These have become more and more sophisticated over decades of research and development and today already have high standards in robustness and efficiency. Much more, deep neural networks can coexist in harmony with classical solution methods. Their development without elaborate modeling and simplification, but as data-driven approaches, can pave other paths of scientific computing. It has to be emphasized that the presented results are promising, but on the other hand, they raise more questions than they answer. The whole process in the development of neural networks is not to go by standard approaches and theoretical proofs for the optimality of certain methods are missing. In the experiments performed, it has been repeatedly found that a particular network architecture works very well for one particular partial differential equation, while it is useless for other partial differential equations. Thus, the development of equation solvers for partial (non)linear differential equations is still in its infancy but has the potential to enrich both fields and lead to highly effective developments.

A summary of machine learning in fluid mechanics [BNK20] presents several techniques to understand and model fluid flows in various fields. The advantage of machine learning is clearly to be able to extract information from data, which can be translated into knowledge about the underlying mechanics. It is made clear that besides the enthusiasm about the possibilities of machine learning and the demonstration of successes, the reality must not be lost sight of. Understanding learning algorithms, how they work, and when they succeed or fail is fundamental for further progress in this field and therefore more important than just good results. The application of machine learning to fluid mechanics is an open and challenging field but may open up the possibility of future advances in solution methods such as those achieved in the last century. On the other hand, nature, with its perfectly optimized shapes, surfaces, and motions in both water and air, shows that solving the Navier-Stokes equations should be only one part of improving analyses, learning from examples or paragons another part. Machine learning allows to exploit this second part and avoids the challenges of equation-based analysis such as high dimensions and nonlinearities that prohibit closed-form solutions and real-time optimization. The techniques presented cover a variety of benefits and challenges: From measurements that are nondeterministic, such as particle transport in a fluid flow, to robust parameter selection in Bayesian inference from a probability distribution, to denoising algorithms with imaging science methods. Fluid mechanics, with its sensitivity to noise, disturbances, latent variables, and transition states, has many more pitfalls than, for example, pattern recognition in images. What would be desired are physically interpretable and generalizable models, but there is still a long way to go before this can be achieved. Currently, the focus in fluid mechanics is on hybrid models that combine data-driven and physically derived approaches; the potential application areas are large and many possibilities have yet to be discovered.

Models of fluid-structure interaction in pipe flows are presented in [FMSC18] and give an overview of their description. Diverse arrangements and occurring degrees of freedom are presented and their systems of equations are set up as well as conventional couplings are described. In particular, the models also include angles and valves at which peculiarities in the solution occur and refer to further work in the respective area for a more in-depth analysis. The application of such models to various engineering disciplines is discussed, along with their specific challenges. In particular, it is shown that fluid-structure interaction is a case-dependent problem and no numerical model can describe every experimental setup. In particular, when solving the equations, a suitable coupling of the different degrees of freedom has to be found that does not lead to expensive computations in the end. Similar to neural networks, whose literature will be discussed in more detail in some places in the next chapter, there are no set procedures for coupled fluid-structure interaction problems as to when and how the coupling must be included in the solution. The physics behind these problems is not yet fully understood, which introduces the risk of inappropriate solution methods.

For a deeper look into artificial intelligence theory, especially deep neural networks, [GBC16] as a standard work can provide many answers explaining how different neural networks work. In addition, both the mathematical foundations of machine learning concepts are also discussed here, as well as outlooks on recent research in this large environment.

An insight into the practical implementation of neural networks in Python is given by [Cho17], the content of which was referred to in some places in the practical work.

# 2 Fluid-structure interaction

Physically, a fluid-structure interaction describes the mutual interaction between two media, a fluid and a solid. To describe this interaction, it is necessary to consider both media simultaneously, because the flow of the fluid exerts pressure on the structure on the one hand, while on the other hand, the structure's deformation affects the fluid domain. Depending on the strength of this interaction, the behavior of the system will be fundamentally different compared to the two systems considered separately. An understanding of fluid-structure interaction is important to explain various phenomena in nature and technology. For example, vibrations of structural components surrounded by flow, blood transport through the vascular systems in the body, or the loads on high structures in the wind or in rivers can be described by mutual interactions of the media involved. Depending on the application, there is a one-sided coupling, if essentially one medium is influenced by the other but has no feedback on the other, or a bidirectional if both media exert significant interactions on the coupling partner. For a simulation of a fluid-structure interaction problem, solvers for the fluid equations $F(s) = f$ as a function of the solution of the structural equations as well as solvers for the structural equations $S(f) = s$ as a function of the solution of the fluid equations must be developed and coupled, because the solution in one subdomain can only be computed with the solution from the other subdomain. In most cases, this requires the forces or pressures from the fluid and the displacements of the structure to be passed as coupling data to the other solver. In this thesis, we aim to develop machine learning based solutions for both domains and couple them via a library. The ultimate goal, which is beyond the scope of this work, is to use these surrogate solvers to improve the efficiency and performance of classical solvers.

## 2.1 Numerical solution methods

Fluid-structure interaction problems are usually investigated by means of classical numerical solutions. For this purpose, systems of equations describing the problem must be created, the resulting partial differential equations must be discretized in time and space, and then the solution of all equations at all grid points must be determined iteratively for each time step, taking into account the current solution of the equations in the other domain. There are two different approaches how this can be done: Either one large system of equations is set up for both parts (monolithic approach) and this is solved jointly, or the problem is split into two coupled subproblems (partitioned approach) which depend on each other and are solved iteratively.

In this thesis, we follow a partitioned approach, in which the individual subproblems are to be solved independently and then exchange data to account for mutual coupling. This requires suitable program structures that control the communication and monitor the progress of the individual solvers. The advantage is obvious: The individual subproblems can be solved in parallel, changes in one subproblem can be implemented independently of the other, the code becomes clearer, and the resource requirements for each subproblem are considerably reduced compared to the former approaches. Accordingly, such approaches are to be preferred whenever possible.

The most widely used discretization methods are finite elements, finite differences, and finite volumes. A finite element discretization divides a body with unknown behavior into many small, simple sub-bodies (finite elements) with easily computable behavior. Here, the solutions to the differential equations are usually calculated at the corners, called nodes, of the finite elements and then interpolated to the entire element, yielding continuous solutions.
In finite difference methods, all derivatives occurring in the (partial) differential equations are approximated by difference quotients. As with finite element methods, this requires a grid on which the discretized derivatives can be determined and solved. Boundary and initial value conditions can of course be specified and usually also fulfilled.
Finite volume methods solve (partial) differential equations which are based on a conservation law. The computational domain is divided into arbitrary volumes and the solutions of the equations are calculated in the centers of the individual finite volumes. Also, transition conditions can be defined at the boundaries.

For the solution of such problems a large variety of commercial and open-source software exists. Mutual coupling can usually be defined implicitly or explicitly. In the case of implicit coupling, many iterations of the solver are performed until the solution for a time step has converged or another termination criteria has been reached. This allows strongly coupled problems to be simulated over long periods. Explicit solvers, on the other hand, perform only a single iteration per time step. This usually requires less computation time, but strongly coupled problems are often not solved with sufficient accuracy and time steps have to be chosen very small. Great amount of research has been performed in this field, however, since in this thesis no numerical solution methods are developed, but (partial) differential equations are to be solved by machine learning techniques, a detailed treatment of the numerical aspects is omitted at this point. However, since in most cases no analytical solutions exist for coupled fluid-structure interaction problems, numerical solutions published in the given literature or computed with commercial software are used as reference solutions for the neural networks.

# 3 Neural Networks

Since modern computers have become more and more capable, more and more different tasks can be solved efficiently with them. In recent years, one topic, in particular, has taken center stage: artificial intelligence. The dream that computers can take over the tedious parts of human work is as old as computers themselves, and now it is even being expanded: computers are supposed to independently gain new knowledge from existing data. To let this dream become reality, so-called artificial neuronal networks have been and are being developed, which attempt to mathematically describe and reproduce thought processes in the human brain. The progress in storing and processing large amounts of data has been enormous in recent years, so that, with the help of the right algorithms, this dream is more and more taking shape.

Artificial intelligence has already mastered many tasks with flying colors. It has been able to develop extremely good opponents in certain games such as Go, reliably read characters in pictures, track and classify objects in videos, understand natural language, and much more. Many boundaries of what is possible through artificial intelligence are constantly being pushed, and new areas where artificial intelligence shows its ability to solve complicated problems are being opened up. Thus, a variety of different types of artificial intelligence have recently been established, all of which work particularly well in different application areas and are combined differently depending on the case. The abundance of specializations and variations that now exist is becoming increasingly difficult to summarize, making it all the more important, on the one hand, to identify the basic types that are suitable for a particular task and to combine them in a meaningful way, and, on the other hand, to find modeling options that are even closer to the human model and that can better describe and replicate the thought processes in the brain.

In this chapter, the properties of neural networks that are suitable or required for the solution of a coupled fluid-structure interaction problem will be considered and explained in more detail.

## 3.1 Machine learning techniques

Each approach of machine learning techniques has its prerequisites and capabilities. The methods which are used to approximate the function f(x)=y are called predictive models, which can be categorized as classification or regression, depending on the output variables. In classification, the output is a discrete value that can be assigned to a specific category, such as: "There is a person in the picture" or "The voice is friendly." If clustering, which is unsupervised classification, is performed, the result is also a discrete value but has no class label. Only similar samples are given the same label. In regression, there is a continuous target value such as a real function value when a function is approximated. Machine learning techniques can be classified into different categories. Approaches range from untrained to elaborately trained models. The former, such as k-nearest-neighbor, are ready to use, but often have complex computations when classifying a test

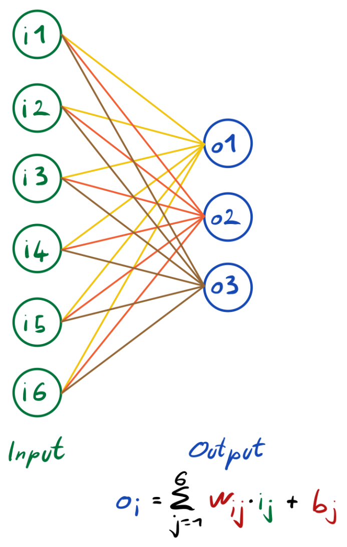


**Figure 3.1:** Fully connected neural network with 6 input and 3 output neurons. Each input is connected to each output with a weight. In addition, each output neuron can learn a bias.

sample, while the latter, such as a neural network, involve a lot of offline work, but online, when classifying or regressing a test sample, consist only of a series of simple mathematical operations and are therefore very fast. For the problem to be solved in this thesis, where complex relationships between fluid and structure are to be learned and continuous target values are required, i.e. a regression is to be performed, neural networks are the first choice.

## 3.2 Fully connected Neural Network

Fully connected neural networks connect all inputs with all outputs. The output $o_j$ of a neuron $j$ is calculated according to

$$o_j = (\sum_{k \in \boldsymbol{i}} i_k W_{kj}) + b_j,$$

with an input vector $\boldsymbol{i}$, a weight matrix $\boldsymbol{W}$ and a bias vector $\boldsymbol{b}$. i.e., a linear regression is performed between the data. This can also be compactly stated in matrix notation for $n$ inputs and $m$ outputs for a complete layer as

$$\boldsymbol{o} = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b},$$

where $\boldsymbol{o}$ stands for the output vector of length $m$, $\boldsymbol{W}$ for the $m \times n$ weight matrix with entries $W_{ij}$ for the weight between output $i$ and input $j$, $\boldsymbol{x}$ for the input vector of length $n$ as well as $\boldsymbol{b}$ for the bias vector of length $m$.
An illustration where the connected neurons are visible is shown in Figure 3.1.

Each output of a fully connected neural network leads to the firing of the corresponding neuron or not through a nonlinear activation function. This is important because otherwise the concatenation of multiple fully connected layers would still be mathematically a single linear regression and could only learn linear decision boundaries. However, the vast majority of problems lead to nonlinear decision boundaries, which require multiple concatenated layers in a fully connected neural network.

Fully connected neural networks are usually trained by error backpropagating to the individual weights in the training process and updating them along the steepest descent using (stochastic) gradient descent. As can be seen from the definition, a fully connected neural network works for input and output vectors of any length. This makes it particularly flexible to use in multi-layer neural networks. However, it has to be taken into account that for a single layer consisting of $n$ inputs and $m$ outputs $n(m+1)$ weights are introduced into the network, which makes the neural network tend to overfit faster since there are often many more weights to train than training samples.

## 3.3 Convolutional Neural Network

A convolutional neural network generally performs a discrete mathematical convolution between the inputs and some filters, called kernels. Patterns in the input are detected, such as an eye in an image, respectively the similarity between frequencies in the input and the kernel is measured. Mathematically, the convolution $f * g$ of two functions $f, g : \mathbb{R}^n \to \mathbb{C}$ is defined by

$$(f * g)(x) := \int_{\mathbb{R}^n} f(\tau)g(x-\tau)d\tau.$$

In a convolutional neural network, this convolution operation is performed discretely, since both the inputs and the kernels are spatially discrete. In this case, where $f, g : D \to \mathbb{C}$ with a discrete definition domain $D \subseteq \mathbb{Z}$, the one-dimensional convolution simplifies to

$$(f * g)(x) = \sum_{j \in D} f(j)g(x-j).$$

Figuratively speaking, the kernel is moved over the inputs as shown in Figure 3.2 in such a way that it computes the convolution between the local part of the input and the kernel at each admissible point. Convolutional neural networks are modeled based on the human receptive field, in which many photoreceptors on the retina determine the activation of a single subsequent neuron. This reduces the density of information flowing into the brain, making it easier to process signals. In addition, both higher sensitivities are achieved because incoming signals from multiple receptors together stimulate a neuron and, at the same time, insignificant information is filtered out if a neuron is not activated because the receptive field is not sufficiently activated by the input signal. In artificial neural networks, the weights of each kernel are trained using error backpropagation. Different, usually odd, sizes of the kernel can be used. Small kernel sizes (e.g. 3 or 5) extract local patterns in the inputs while large kernel sizes (e.g. 9 or 11) detect more global patterns in the inputs. If a so-called pooling layer is added after a convolutional layer, where out of several neurons only the one with the highest correlation between kernel and input fires, the specificity of the convolutional neural network is usually further improved as the feature space is reduced [LBD+89].

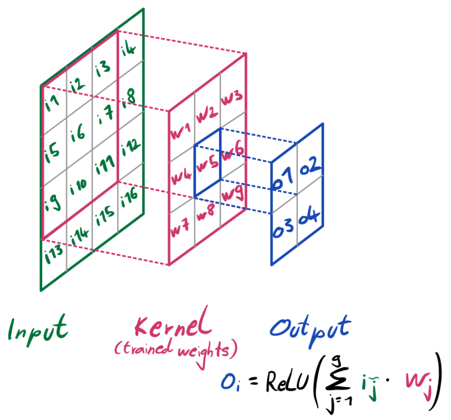


**Figure 3.2:** Simplified operation of a convolutional neural network with ReLU activation function. The kernel (red) is moved to each location where it completely fits the inputs. All weights $w_j$ are learned during training.

## 3.4 Long short-term memory

Recurrent neural networks use the outputs of neurons in a given layer at a given time to make decisions in the next time step based on previous decisions in the same layer or earlier in the neural network. This can be used to detect connections in input sequences, for example, to determine matching words to a sentence that has been started. They are a powerful tool in learning (time-) dependent states, but, as shown in [Hoc91], they have the disadvantage that during error backpropagation, due to vanishing gradients in the front lying layers, only the rearmost layers of the recurrent neural network are effectively trained. To prevent this, recurrent neural networks have been improved by introducing gates so that the derivatives of the errors are approximately equal over the entire network. This new version is called long short-term memory because it can remember experiences far from the past. Using several gates, a cell state, and a hidden state, which is the output of the cell in the last time step, it can remember how data was related in the past and tries to extrapolate these relationships into the next time step. Long short-term memory neural networks are therefore particularly often used to predict time series or temporal relationships in data.

## 3.5 Deep vs. shallow Neural Network

At the beginning of the development of artificial neural networks, they mostly consisted of only one hidden, fully connected layer with a nonlinear activation function. According to the Universal Approximation Theorem, first introduced in [Cyb89], already a neural network with a nonlinear activation function and one hidden layer can approximate any continuous function $f : \mathbb{R}^d \to \mathbb{R}^D$, with $d, D \in \mathbb{N}^*$, on every compact subset $K$ of $\mathbb{R}^d$ with arbitrary accuracy, so in principle, it can learn anything. But on the other hand, deep neural networks often show much better results in experience than shallow neural networks. What this is due to, and in particular how important it is for artificial intelligence to be analogous to the human model, requires further research. Since neural networks can be effectively trained on graphics processing units (GPUs), deep networks have largely prevailed over shallow networks. More layers provide more opportunities to analyze different aspects of the task being solved. Generally, neural networks are initialized with random weights and after the forward pass of each minibatch

$$\hat{\boldsymbol{y}}(\boldsymbol{x}) = \boldsymbol{W}^{(2)} \left(\sigma\big(\boldsymbol{W}^{(1)}\boldsymbol{x} + \boldsymbol{b}^{(1)}\big)\right) + \boldsymbol{b}^{(2)},$$

here demonstrated for a two-layered fully connected neural network with sigmoidal activation function after the first layer and identity activation function after the second layer, a given loss function is used to calculate the error between the predicted and the actual target.
With the definition

$$\begin{aligned}
\boldsymbol{x}^{(0)} &= \boldsymbol{x} \\
\boldsymbol{a}^{(l)} &= \boldsymbol{W}^{(l)}\boldsymbol{x}^{(l-1)} + \boldsymbol{b}^{(l)}, \quad l = 1,2 \\
\boldsymbol{x}^{(1)} &= \sigma(\boldsymbol{a}^{(1)}) \\
\hat{\boldsymbol{y}} &= \boldsymbol{a}^{(2)}
\end{aligned}$$

this forward pass can also be rewritten as

$$\hat{\boldsymbol{y}}(\boldsymbol{x}) = \boldsymbol{a}^{(2)} \circ \sigma \circ \boldsymbol{a}^{(1)}(\boldsymbol{x}^{(0)}).$$

The mean squared error loss is then calculated according to

$$L^{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) := \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$$

and the chain rule is applied to get the partial derivative of the loss function with respect to each weight or trainable parameter in the neural network:

$$\frac{\partial L}{\partial w^{(k)}} = \frac{\partial L}{\partial a^{(l)}}\frac{\partial a^{(l)}}{\partial x^{(l-1)}}\frac{\partial x^{(l-1)}}{\partial a^{(l-1)}}\frac{\partial a^{(l-1)}}{\partial x^{(l-2)}}\frac{\partial x^{(l-2)}}{\partial a^{(l-2)}}\cdots\frac{\partial a^{(k+1)}}{\partial x^{(k)}}\frac{\partial x^{(k)}}{\partial w^{(k)}}$$

Here, $w^{(k)}$ is a weight in the $k$-th layer of the neural network with in total $l$ layers. Subsequently, all such parameters are updated by an optimizer such that, figuratively speaking, each weight is adjusted a fixed increment along the steepest descent of the error function. One problem with deep neural networks are often vanishing or exploding gradients in the error backpropagation. Applying the chain rule across all layers and activation functions of a neural network often results in products of many different factors. Here, most common activation functions have derivatives between 0 and 1. If these are often multiplied for far forward layers, the product is usually vanishingly small, so that updating the weights has almost no learning effect. On the other hand, long products of factors greater than one lead to explosively high derivatives and a small update step run completely over the target. To avoid this problem, recurrent neural networks have been replaced by long short-term memory layers, which allow a nearly constant error gradient over many layers, and sigmoidal activation functions have been replaced by Rectified linear units (ReLU), which have either an error gradient of 1 when the corresponding neuron is active or 0 when it is deactivated and thus not trained. Thanks to these improvements, multi-layered deep neural networks are well trainable and often significantly outperform shallow neural networks.

## 3.6 Hyper parameters of Neural Networks

Each artificial neural network can be seen as a program with properties defined during programming. After defining the program code, all trainable parameters are trained using training data in a way such that the resulting error is minimized. The architecture of the neural network thereby always remains the same. However, many properties and capabilities of a neural network depend directly on the architecture of the neural network. For example, the type, number, and order of the different layers, the number of hidden neurons, their activation function, the loss function used, the step size of the optimizer and the optimizer itself, the batch size, the initialization of the weights, the amount of training data and several other factors play a significant role in determining how powerful a neural network actually is in the end. If the neural network has too few parameters, it cannot learn complicated problems; if there are too many, the training data will not be sufficient to train each weight expediently and the neural network will overfit to the training data. Likewise, small sizes of minibatches lead to inefficient training while minibatches that are too large cannot be fully loaded into the GPU's memory and must be swapped to main memory or unexpectedly abort training. Similar problems can occur for all other hyperparameters and must therefore be recognized and corrected during programming in order to achieve good results. In the following, two important aspects of the hyperparameters of a neural network will be examined in more detail.

### 3.6.1 Optimizer

In order to train the weights introduced in a neural network, a loss function is needed on the one hand, which quantitatively specifies how large the discrepancy between the predicted and the actual target is, and on the other hand, an optimizer that adjusts the weights in such a way that the loss function is minimized for all training samples. This optimizer has to be given as a hyperparameter of the neural network and therefore has to be well chosen. So-called gradient-based optimizers are often used, which compute the partial derivatives of the loss function with respect to each weight and then walk a fixed step size along the steepest descent to determine the new weight. This update of the weights can be done either after each training sample, after each minibatch, or after each complete run of the training data set according to the rule

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \alpha \nabla L(\boldsymbol{\theta}_t, \boldsymbol{x}, \boldsymbol{y})$$

where $\theta_t$ are the weights of the neural network at the t-th pass, $\alpha$ the step size and $L(\boldsymbol{\theta}_t, \boldsymbol{x}, \boldsymbol{y})$ the loss function depending on the weights and the training samples passed and their target values $(\boldsymbol{x}, \boldsymbol{y})$. Such (stochastic) gradient descent methods thus converge exactly when the gradient $\nabla L$ of the loss function vanishes. This is geometrically the case for high, low, and saddle points of the loss function. It is practically impossible that the optimizer converges to a high point because it always runs along the steepest descent. However, it may converge at saddle points or local minima and thus the global best solution is not found. To avoid reaching saddle points and converge faster towards a minimum, the so-called Adaptive moment (Adam) optimizer was introduced in [KB17], which in addition to descending along the gradient, takes a step of adaptive step length along the direction of the last update. Empirically, Adam shows better results in practice than other stochastic optimization methods.

### 3.6.2 Hyper parameters optimization

Above, it has already been described how hyperparameters affect a neural network; to ensure that it does not suffer from model errors, hyperparameters must be optimized. This is usually done by simple trial and error, i.e. changing individual hyperparameters while the rest of the model remains unchanged. In the end, those values are adopted which reduce the error as much as possible and still provide a robust result, i.e. are less sensitive to stochastic fluctuations. There are no generally valid recommendations for hyperparameters, they strongly are dependent on problem to problem and network architecture and therefore have to be optimized for each neural network. This step is very time-consuming since a neural network often has to be trained for several epochs to determine the influence of a specific hyperparameter, which may additionally depend on other hyperparameters. Nevertheless, a neural network with wrongly chosen hyperparameters can lose its generalization ability and thus become useless. This can usually be detected in a large difference between training and test error rates.

## 3.7 Training of Neural Networks

A neural network can only execute what it has learned. Simply creating a suitable architecture does not provide information on how inputs and desired outputs are related. Therefore, the weights of each neural network must be trained so that these relationships can be recognized and processed in the data. Usually, the weights are initialized randomly or with zeros and then trained by first running a minibatch of training samples through the neural network, propagating the resulting errors back to each weight, and then updating the weights one step in the direction of the steepest descent of the loss function. This is repeated for all minibatches in the training data set so that by the end the neural network has seen each training sample once and the weights have been adjusted to reduce the error. Of course, it is possible that there are more trainable weights in the neural network than there are training samples, i.e., an underdetermined system of equations is to be solved. In this case, depending on the architecture, the neural network may simply create its own path for each training sample without learning any correlations in the data. In addition, it is quite likely that after running all the minibatches, the weights have not yet converged to a certain value and thus a minimum has not yet been reached. Therefore, this process, called a (training) epoch, is repeated several times until a good approximation to the minimum is achieved. However, this does not directly solve the first problem and it must be solved by other methods, see section 3.7.2.

Such a training is computationally very expensive. Many derivatives must be computed, stored, and weights updated, inputs must often be scaled and rescaled, and large amounts of data must be copied between main and working memory. Other training methods that are less computationally intensive usually show much poorer generalization capabilities. This is also why neural networks have long been limited to a few layers with relatively few hidden neurons. However, since tremendous progress has been made in the computational power of new CPUs, increasingly complicated neural networks can be trained in a reasonable amount of time. In addition, neural networks can be trained much more effectively on GPUs with appropriate software, which ultimately enabled the breakthrough in deep neural network training.

During training, it must be determined how many epochs the training data set should run through the neural network, if it is trained too little, it is underfitted to the task, if it is trained too much, it is overfitted to the task and likewise not optimal. Therefore, it must always be appropriately evaluated when to stop training, perhaps with a validation data set that runs through the neural network every $n$ epochs and shows the current error rate for unseen samples. Then it is stopped early, as soon as the error rate does not decrease in the medium term.

### 3.7.1 Data preprocessing

In order to train or apply a neural network, the data must be prepared in the correct form. Depending on the data set, the individual files must be loaded and preprocessed so that they all have the same form because the number of input neurons is mostly fixed by the architecture and cannot be changed from sample to sample. Preprocessing includes many steps that may or may not need to be applied depending on the data set, such as

- splitting into training and test dataset,
- scaling of each feature to the same range of values,
- transforming data of different shapes into inputs of the same shape,
- generating new training data from the existing ones, by scaling, deforming, rotating, or similar to get a sufficiently large data set,
- converting class labels into ordinal or cardinal scales, respectively and
- identification of outliers that will not be included in the actual dataset.

All of these steps are designed to allow the neural network to work with the data, learning the actual relationships between the data as well as possible. This prevents it from later learning statistical artifacts that would lead to invalid results in the application.

### 3.7.2 Overfitting

A common cause of a large discrepancy between errors in the training and test data sets is that the neural network is overfitting to the training data and has effectively stored them without actually learning why such input should lead to a certain output. This effect is called overfitting. To avoid overfitting, regularization techniques are used during training to prevent a sample from taking exactly the same path through the neural network on each run. Dropout is particularly effective in this regard, where in each run of a minibatch, each hidden neuron is dropped out with a probability of $p_{Dropout}$ and thus is not trained. This requires the same training sample to take a different path through the neural network on each run without increasing the resulting error, thus the neural network is forced to actually learn why a particular input is assigned to a particular target. Furthermore, the data set can be extended so that the same sample does not pass through the neural network multiple times during the training process using data augmentation, within the limits of what is physically allowed or what makes sense from an application perspective. This is especially useful when there are only small amounts of labeled samples and the labels do not change as the data changes, as is often the case when classifying objects on images. For example, a mirrored, rotated and black and white object will always remain the same object as in original and in color. If the data itself cannot be changed without violating the physical properties of the problem, some noise can be added to the data. Due to measurement errors, the data set usually does not have a 100 percent match to reality, which can justify this approach. Adding some noise can also be extended to the hidden neurons by randomly giving their weights small changes after each run. Finally, early stopping should be mentioned as an effective regularization method. The error on the training data set tends to decrease monotonically the longer the neural network is trained. At the same time, the error on the validation or test data set reaches a minimum and then increases again. This indicates overfitting to the training data. If the model is trained only as long as an improvement on the validation or test data set can be detected, another cause of overfitting is avoided.

### 3.7.3 Re-use the Prediction

Neural networks almost always need to be tailored exactly to the problem being solved. For the most part, they can solve one task very well but are completely incapable of all other tasks. If a neural network is trained first on one task and then on another task, it may forget the knowledge of how to solve the first task. In order to solve partial differential equations, it must be specified beforehand exactly what belongs to the task to be solved by the neural network and what does not. It can already be specified at this point that starting from the last $m$ time steps $t_{n-m+1}, \dots, t_n$, with $m > 0$, the neural network should determine the solution of the partial differential equation at the next time step $t_{n+1}$. At the same time, predicting for only always the one next time step would be ineffective, so the predicted solution is supposed to be iteratively reused to determine the solution at time step $t_{n+2}$, using the solutions at time steps $t_{n-m+2}, \dots, t_{n+1}$. Here, the inputs to the time steps $t_{n-m+2}, \dots, t_n$ are the „true“solutions at these time steps and the solution belonging to the time step $t_{n+1}$ is the one predicted by the neural network. Thus, the task is predicting the solution for any number of time steps in the future. For this, the neural network should also be trained in such a way that it must continue to use its own predictions to determine the solution in time steps further in the future. This should especially help to obtain „stable“solutions. Nevertheless, it must already be stated at this point that machine learning techniques comparable to this task cannot provide 100 percent correctness. Thus, it would be wrong to expect that lower errors can be obtained by similar methods for a coupled and thus more difficult problem. Nevertheless, by reusing its own predictions, it will later become possible to obtain a stable solution over several time steps in the future. However, since the expected correctness is lower than it would be necessary for a physically correct description of the system, later, after every or every x-th time step, a numerical solver could compute the correct solution of the partial differential equations using the quickly determined prediction of the neural network as an initial estimate. This is precisely the significant advantage of a neural network over numerical solvers: they are incredibly fast in their prediction and could thus reduce computationally intensive iterations of the numerical solver. But more about this will come at section 4.3.

### 3.7.4 Validation of the model

In the end, the computation of a solution of the neural network is very difficult or even impossible to trace. To make conclusions, where and why errors occur, is an enormous effort. The neural network learns the connections between the data independently, which has, on the one hand, the advantage that, with appropriate measured data, no modeling errors arise by unknown factors, on the other hand, the disadvantage that a neural network does not describe, why it learned exactly this connection and whereby this can be physically justified. This is a critical point especially in a simulation: if it is unknown why a model behaves the way it does, how can it be verified? A simulation model, which is not validated for the different partial aspects by simpler experiments or something comparable, cannot be accepted or at most only with great caution. Each simulation usually provides a result; if it can only be determined later by the consequences of conclusions drawn from it that the model was faulty, great damage may already have been caused! In order to be able to provide reliable results, all neural networks used in this work are validated on test data sets, knowing well that only the results are somewhat little to validate a model; however, even more comprehensive validation experiments would go beyond the scope of this work. Given the fact, that the focus here is on extending machine learning techniques to solving coupled partial differential equations, and that it is not yet possible to say with certainty whether this actually succeeds, this type of validation seems to be sufficient for the moment.

# 4 Machine learning for Partial Differential Equations

Solving partial differential equations has always been a difficult task. While many solvers are relatively easy to implement, iterating over each grid point takes a lot of time. Especially small discretization widths or high dimensional equations are often a time-consuming endeavor. Accordingly, there is a great effort to find new solution methods that can ease this task. For example, in [RPK19], neural networks were presented that determine parameters in partial differential equations while showing very low errors. In recent years, several different ways to solve partial differential equations using machine learning techniques have already been developed, almost all publications on them use deep learning for this complex task and require fast computers or a lot of computational effort in the training process. The results are already very good, with innovative algorithms very low errors have been produced for many test cases. The goal of this thesis is to explore the possibilities of a new approach: In a first step, using the solutions of the last time steps for uncoupled structural or fluid equations by neural networks, a solution for the next time steps of these equations shall be obtained. In a second step, these uncoupled neural networks are to be coupled with each other in order to determine the solutions for the coupled equations using the same approach.

## 4.1 How powerful machine learning is

Machine learning approaches are not particularly new today. Since the 1940s, the first simple linear neurons were developed that could simply learn linear tasks. Later on, nonlinear neurons and error backpropagation were added as learning algorithms, which enabled the learning of simple nonlinear tasks. After little relevant progress was achieved between 1990 - 2010, the introduction of deep neural networks and effective regularization methods since 2010 also enabled the solution of difficult problems as found in real-world applications. At the same time, large amounts of data (Big data) could be stored and processed in computers, enabling data-based approaches. These techniques enable us to learn complex behaviors from merely looking into the data.

The fact is, artificial intelligence is currently developing very fast. New methods of training neural networks more effectively are being developed continuously and more available computing power, as well as data storage, enables the application of machine learning to all areas where data can be produced cheaply. Results are currently not always perfect, but processing very large amounts of data already shows in many areas that correlations can be identified which are difficult to find using other methods.

## 4.2 Frameworks for machine learning

To be able to implement a neural network, a suitable framework is needed to run the program on a computer. A framework that is as widely used as possible and that supports graphics processor training shall be used here. Keras, TensorFlow, and PyTorch are available for this purpose, which will be examined here in more detail for their advantages and disadvantages in order to be able to select a suitable framework.

Keras is an open-source library written in Python. It was developed to perform fast experiments with deep neural networks. Therefore it has a high-level application programming interface, which allows fast development of code. For high-performance applications, Keras is less suitable, as it is designed more for usability than for performance optimization. Since Keras models are mostly simple, there are few needs and possibilities to edit code when errors occur. It is mostly applied only for small datasets, as the longer execution time due to the high-level application programming interface makes large datasets ineffective to process.

TensorFlow is also an open-source library. It was released in 2015 and offers both high- and low-level application programming interfaces, which made it a widely used framework. It enables fast executions of code and graphics processor training through Nvidia CUDA, which becomes a particular advantage with deep neural networks. Debugging a program usually turns out to be very difficult, even if it is possible. The good performance of TensorFlow makes it well suited for neural networks that require fast execution. There is extensive documentation and many ways to visualize results. A clear advantage is that trained models can easily be used in actual applications since there is a special framework called the TensorFlow Serving framework [ABC+16].

PyTorch, which was released in 2017, is just like the other two an open-source machine learning library that has a lower-level application programming interface, which allows direct changes in tensors but in the style of Python allows fast implementation of code at the same time. It has high performance and many ways to debug code and find errors. The Nvidia CUDA library enables to effectively train the models on graphics processors. It also handles large datasets well and allows efficient execution of the code. It can be used quite flexibly and has a high similarity to other Python functions or libraries, which means that many parts of a PyTorch model can be understood even with Python knowledge. There is extensive documentation of all functions and a large community that can find or has already found good solutions to problems that arise. It supports parallel execution just like Python.

Keras could be excluded from this selection because a simple implementation is not the most important criterion and models developed here should also have the possibility to be extended later. Tensorflow could implement the neural networks to be developed here well, but the possibilities of PyTorch and its very Python-related style form the best overall package here for developing neural networks and later coupling them together as well.
Therefore, the practical part of this work will be in Python using the PyTorch framework.

## 4.3 Limits of classical numerical solution methods

Classical solvers for solving (partial) differential equations are already well-developed in many aspects and provide solutions that in principle converge to the actual, mathematical or physical solution within machine accuracy. Nevertheless, this accuracy is not the only criterion by which a solution method is evaluated. A whole pallet full of characteristics should be fulfilled in order to receive an optimal solution. Thus the procedure should be [HK20]

- consistent, meaning that for smaller and smaller time and location step sizes, the discretized equations become the original (partial) differential equations,
- monotone, which means that the solution does not generate new extrema, thus preventing oscillations and unphysical solutions,
- stable, which means that the solution does not explode even for imprecise inputs, so rounding errors do not add up,
- efficient, which means that the solution is found with as little resources as possible, as already mentioned
- convergent, which means that the discretization error tends to zero for smaller step sizes, i.e. the difference between the obtained and the actual solution tends to zero, and additionally
- of high order $p$, which means that the discretization error is of order $O(h^p)$, with step size $h$.

For this, the problem must additionally be well-conditioned, i.e., for smaller and smaller perturbations in the input data, the error between the actual solution of the (partial) differential equation with these perturbations and the actual solution for unperturbed input data should vanish. This must happen faster than the norm of the error disappears. In some cases, these properties are also mutually exclusive and no known method exists yet that is optimal for all properties. In addition, there are limitations in the hardware. Storage options are limited and energy needed for the solution costs money and pollutes the environment. Furthermore, each iteration over many grid points takes a corresponding amount of time.

Despite advancing possibilities due to fast computers and improved algorithms, classical numerical solution methods are limited. A compromise between accuracy, speed, and the other properties must always be found. Especially when the problem requires small time steps and long simulation times are important, many iterations of a solution procedure are performed.
At this point, machine learning techniques can contribute to solve problems faster: Neural networks are usually extremely fast in determining the solution for given inputs since a large part of the computational effort is put into the offline phase during development in order to then only have to fall back on learned correlations in the online phase in the field. The computations required to solve a time step can therefore be greatly reduced. The possibilities, challenges, and limitations for such machine learning approaches will be considered in more detail in the practical part of this thesis. Since the ultimate goal is on the one hand a data-based approach, which on the other hand should be faster than the standard numerical methods, it is clear that in this initial phase of the use of machine learning techniques for solving coupled (partial) differential equations, the focus will be more on the data-based part. Optimizations in efficiency only make sense when working methods have been found. Comparisons about training or computing times are therefore to be evaluated under this aspect.

# 5 Problem definition

With this background, we start defining the concrete problem definition for the practical part of this thesis. In many publications the potential of machine-learning approaches for the solution of partial differential equations has already been shown. Especially neural networks could show their ability to correctly identify parameters in partial differential equations in [RPK19]. Data-based approaches are particularly convincing due to their ability to get along without classical modeling. Thus, physical systems can be described without solving large (non)linear systems of partial differential equations. The resulting models do not suffer directly from simplifications and assumptions, which are made in the modeling process. Due to their construction, they are also less endangered to neglect unimportant or even completely unknown relations, although they would be of crucial importance in certain aspects. For all these advantages, the established modeling ways have to be abandoned for this purpose. At this stage, the new model will not be able to describe the individual physical backgrounds of the learned effects in a way that can be understood by humans. Instead, given a neural network architecture to be defined at the beginning of the practical part, it will process data in such a way that the desired output data are determined from specified input data. These have a physical origin and a meaning, which the neural network should learn independently and transfer to unknown data.
However, the goal here is not to determine parameters in partial differential equations, as this would again contradict the data-based approach without a classical model. Instead, the local solution or the solution in the whole computational domain shall be transferred to the neural network in the last time steps to predict the solution in the next time step. This approach is to be extended later in such a way that this prediction can be continued iteratively in order to be able to predict the solution of time steps lying further in the future. This approach shall later have the flexibility to be transferred to other potential application areas if the results in the practical part will confirm this design of neural networks as a reasonable choice.

We do not seek to replace classical solvers of ordinary or partial differential equations with neural networks. The solutions of this first practical part shall be further improved in order to subsequently be able to find solutions for coupled fluid-structure interaction problems in addition to the solutions for fluid and structural mechanics problems. To solve the resulting coupled partial differential equations, neural networks for the sub-domains for the fluid domain and the structural domain will be trained according to the physics involved. These single-physics solvers for fluid and structure are coupled with the coupling library preCICE [BLG+16] to obtain a multi-physics solver for predicting the solutions of the coupled partial differential equations. In each case, a single-physics solver specifies the boundary values of the other single-physics solver, which are exchanged with each other through the coupling library. Research in similar areas has shown that the expected accuracy in the obtained solution is lower than it is possible with classical numerical solution methods. To deal with this, the neural networks will be implemented in such a way that it is possible to incorporate them into classical solution procedures, so that the neural network predicts an initial solution and the numerical solution procedures improve it with fewer iterations. The results are intended to show the extent to which this data-based approach has the potential to improve the effectiveness of equation solvers and should be investigated further in the future.

# Part II: Bachelor thesis

After all theoretical backgrounds have been explained, in this second part, the neural-network-based coupled framework is implemented to investigate its accuracy and performance. For the implementation, the machine learning framework PyTorch is used, which was evaluated in the previous part as the most suitable for this purpose.

Overall, this practical part consists of many smaller sub-parts, each with its contribution to the larger project. For each of these sub-parts, a suitable test case will be defined at the beginning, which can well represent the goal to be achieved with it and can be adopted in a similar way for later sub-parts or inspire them.

For each sub-aspect of the entire project, a separate neural network must be developed, which is tailored to the specific task to be solved and can learn the respective physics of the problem.

For each neural network developed, data sets are searched for or created at the beginning, which provide the solutions of the differential equations searched for in a suitable discretization of the problem. These reference solutions are computed by classical numerical solvers and can be presumed to be sufficiently accurate.

The data sets are imported for the respective neural network and all contained data are prepared in such a way that they are suitable for a run through the neural network and the target values, i.e. the solutions in the next time step, are stored appropriately. Each data set is divided into a training and a testing data set, which is used for training and testing the neural network, respectively.

At the same time, for each problem, a suitable network architecture is needed, which can learn the physics behind the data. This is done by trial and error to determine whether the selected architecture, meaning the individual layers, their arrangement, and parameters, is suitable for learning a specific problem. Furthermore, other hyperparameters, which later directly influence the quality of the final neural network have to be determined appropriately.

In order for each neural network to actually learn the important relationships in the data, it will be trained with the training data set after initialization. During the training process, the learning progress will be monitored regularly and, if necessary, the configuration will be adjusted to make learning more effective. Each fully trained model will be validated using the test dataset to produce an evaluation of how well it performs the actually given task.

Finally, for each neural network, the obtained results will be presented with the help of visualizations of the outputs, in order to investigate the inference accuracy.

# 6 Single-physics neural network solvers

Before the numerical solver for a coupled fluid-structure interaction problem can be replaced, reliable solvers for uncoupled problems must be developed. For this purpose, we develop neural networks to investigate a one-dimensional beam and a Kármán vortex street. It will be shown that, in addition to numerical solvers, neural networks can also make good predictions for solving ordinary and partial differential equations, respectively. For this purpose, solvers will be developed which can use the data of past time steps, here called history, to determine the desired solution in the next time step. This will be done in an attempt to obtain a model that keeps the solution stable not only for the next time step, but as far into the future as possible, and with sufficient accuracy to capture the physics behind the differential equations. Having developed solvers for both the structural and fluid domains that meet these requirements, these solvers will be further developed towards a coupled problem so that the preCICE coupling library can also be used to solve coupled partial differential equations. For this purpose, the developed solvers will be adapted and improved, so that in each case one solver is responsible for one domain and receives the solution of the coupled problem as input value at the last time steps and the solution of the other solver at the current time step as boundary conditions and thus predicts the solution at the next time step. The coupling will run according to an explicit and an implicit coupling scheme so that it can be iterated until the solution converges. This also allows the extension to strongly coupled problems, which, however, is beyond the scope of this thesis.

## 6.1 1-dimensional Beam

In order to implement an equation solver for structural deformations, a suitable test case is needed. In the following, a fourth-order partial differential equation is to be solved using the example of a one-dimensional beam clamped on one side and vibrated by a load at the free end. Thereby, vibrations of the beam at its lowest natural frequency will be predicted.

### 6.1.1 Euler-Bernoulli equation

For the derivation of an analytically solvable equation, the following simplifying Bernoullian assumptions are made, which are valid for the considered beam with its properties (length, profile, homogeneous mass distribution, characteristics...) in good approximation:

1. The beam is thin: its length is much greater than its cross-sectional dimensions.
2. Beam cross-sections that were orthogonal to the beam axis before deformation are also orthogonal to the deformed beam axis after deformation.
3. Cross-sections remain inherently plane after deformation.

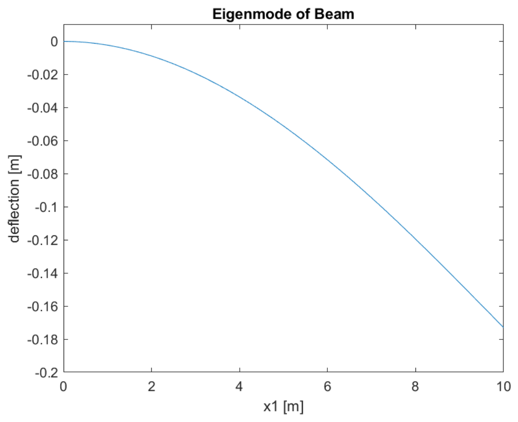


**Figure 6.1:** Schematic illustration of the vibration of the beam with clamped left end and free right end with an external force at the end point.

4. Bending deformations are small compared to the length of the beam.
5. The beam is made of isotropic material and follows Hooke's law [BC09].

This results in the following partial differential equation:

$$EI\left(\frac{\partial^4 \omega}{\partial x^4}\right) = -\mu \frac{\partial^2 \omega}{\partial t^2} + q(x)$$

with the mass per unit length $\mu = \rho b h \cdot 1m$ $[\frac{kg}{m}]$. In this, $E$ is the modulus of elasticity, $I$ is the axial area moment of inertia, $\omega$ is the displacement, $\rho$ is the density, $b$ is the width, and $h$ is the height of the cross-section.

The Euler-Bernoulli equation can be solved both analytically and numerically. It is considered a standard case in the solution of beam-theoretic problems and the correctness of a solution obtained can be easily established by inserting, as is the case for most differential equations.

### 6.1.2 Generation of data

In a data-driven approach, it is essential to provide sufficient data from which the solution can be learned. Since no suitable data set was found in the literature for these purposes, a custom data set was generated for this problem. Different rectangular cross-sections and loads of different magnitude were allowed at the end while the length $l = 10$ $m$, the material (steel, $\rho = 7850$ $\frac{kg}{m^3}$, $\nu = 0.28$, Y-modulus $= 210 \cdot 10^9$ $\frac{N}{m^2}$), the mass distribution (isotropic) and the clamping (left fixed, right free end) remained fixed. To numerically solve the Euler-Bernoulli equation for the configurations included in the data set, the commercial software Abaqus [Aba14] is used, which is a common finite element solver. Here, each beam considered is discretized with a sufficiently tight mesh and, at each time step, the deformation at 41 points of equidistant mesh width between $x_1 = 0$ $m$ and $x_1 = 10$ $m$ ($dx_1 = 0.25$ $m$) is defined as the output variable. The time discretization here is always $dt = 0.5$ $ms$.

In 6.1 a sketch of the setup is shown. According to the support, at the clamping $\omega(x_1 = 0) = 0$ as well as $\omega'(x_1 = 0) = 0$ and at the free end $Q_3(x_1 = 10\ m) = 0$ as well as $M_2(x_1 = 10\ m) = 0$, where $Q_3(x_1)$ is the shear force and $M_3(x_1)$ is the moment curve in the beam. The generated data is saved in a .txt file for each configuration of the beam and then imported for training the model. The final data set consists of eleven different configurations of widths between $0.05\ m$ and $0.3\ m$, heights between $0.1 m$ and $0.3\ m$, and loads between $100\ N$ and $10000\ N$. Each configuration contains as many time steps as needed for one oscillation period.

### 6.1.3 Modeling

Since the equation, and therefore the solution to this equation, is both spatially and temporally dependent, a neural network capable of learning both dependencies is needed. Therefore, the inputs are analyzed by convolution with a convolutional neural network first for spatial patterns and then with a long short-term memory neural network for time-dependent patterns and these dependencies learned from the data in the past are extrapolated into the future. Finally, a layer is needed which makes a prediction of the deformation in the next time step from the processed patterns.
Clearly, the neural network solver should also be able to learn the solution of the Euler-Bernoulli equation for the different cross-sections and loads in the dataset. The model should receive the displacements in the last five time steps (after hyperparameter optimization) for the entire beam as input. This gives an input tensor of size $(5 \times 50)$ (history $\times$ inputs), where the inputs are 9 features of the whole beam + 41 deflections for each gridpoint. The output is to model the deformation of the beam in the next time step. This results in an output tensor of size $(1 \times 41 \times 1)$. As mentioned before, the data are first analyzed for spatial patterns in a convolutional neural network. For this, a 1-dimensional convolution with kernel size 5, one input channel, and 64 output channels is applied, which allows for the detection of diverse patterns between each discretization point and its neighbors. Then, the nonlinear activation function leakyReLU [GBB11] is applied to the data. These hidden features are passed to a two-layer long short-term memory neural network, which then estimates the hidden features in the next time step. Afterward, the non-linear activation function leakyReLU is applied again. Finally, a linear regression of the hidden features is used to predict the deformations at each discretization point in the next time step. It should be noted that in this approach, theoretically, the solution at one grid point may depend on data from distant grid points. This brings the advantage of being able to directly compute the prediction of the displacements at the next time step with a single forward pass and not having to iterate over each grid point.

### 6.1.4 Training

Now the created model has to learn the physics behind the oscillations on its own. To achieve this, the data set consisting of eleven different configurations with 199 time steps each is divided into a training and a test data set. These consist of 75 and 25 percent of the data, respectively, whereby a sample always contains the displacements of the entire beam in the last five time steps and, as a solution, the displacements to the next time step. Here, the training dataset contains only the samples of the time steps at the beginning of the simulation while the test dataset contains only samples of the time steps after those in the training dataset. This ensures the required property that the neural network is actually tested on unknown samples that lie in later time steps than those with which the neural network was trained. The .txt files from the data set are imported into Numpy and processed so that each sample has the described properties. Then the training displacements are fitted with 0.2 percent uncorrelated Gaussian noise to obtain more robust predictions. See more in section 6.1.5. Since it is very important for the training that the data are all in a range that does not lead to either vanishing or exploding gradients, all data are scaled to a range of [0,1] by min-max scalers. For this purpose, a separate scaler is used for the training and test data set, which linearly transforms all deformations into the value range $[min, max]$ according to

$$x_{scaled} = \frac{x - x_{min}}{x_{max} - x_{min}}(max - min) + min \quad \in [min, max].$$

Here $x_{scaled}$ is the transformed deformation in range $[min, max]$, $x$ the original deformation and $x_{min}$ respectively $x_{max}$ the smallest and largest deformation in the dataset.

For training, the Adaptive-moment optimizer, presented in [KB17], is used, which gave better results than Stochastic Gradient descend. An initial learning rate of $lr = 1 \cdot 10^{-3}$ is used. This is automatically adjusted by the optimizer after each epoch. As a loss function to penalize wrong predictions, the mean squared error loss

$$L^{MSE}(\boldsymbol{y}, \hat{\boldsymbol{y}}) := \frac{1}{N}\sum_{k=1}^{N}(\hat{y}_k - y_k)^2$$

is applied. Here $\boldsymbol{y}$ is the true solution, $\hat{\boldsymbol{y}}$ the predicted solution and $N$ the batch size.
Dropout is used with a dropout probability of $p_{Dropout} = 0.3$ on the inputs, after convolution and after the long short-term memory. No dropout is applied before the last linear layer since the network cannot compensate for errors that arise there. A batch size of 1024 is used, which is just large enough for all data to be processed on the GPU. In a first run, the network is trained until a clear tendency to overfitting can be detected. In a further run, this is then prevented by stopping early and the best possible performance is achieved.

### 6.1.5 Validation

To actually determine how well the network has learned the physics, and not just remembered known samples, the trained neural network has to make predictions for the deformations at time steps it has not yet seen, called test data. A discrepancy was found between the errors for unseen and already known data, showing that the trained neural network is overfitting to the training data. To prevent this and to obtain more robust results, different sizes of noise are applied to the training data set in the following, the neural network is trained for 1000 epochs each, and the training errors obtained afterwards are compared against the test errors. When 0.2 percent noise is added to the data, the lowest test errors are achieved.
Furthermore, a simple but very effective regularization method is to apply dropout. This means that in the training process, 30 percent of the neurons are randomly dropped out in each run, and thus the data must be processed over a subnetwork with varying neurons. This avoids that the weights of some neurons are sensitive to the weights of other neurons, which makes it more likely that each neuron does something individually useful [SHK+14]. To find out how large the dropout probability must be for effective training, different dropout probabilities are used for training, the neural network is trained again for 1000 epochs each time with the otherwise unchanged configuration, and the training errors obtained afterward are compared against the test errors. Here, a dropout probability of 30 percent is most effective. In addition, varying the kernel size showed that it should have a

size of five for this test case. At this point, it should also be mentioned that the individual layers of the neural network were not changed in the hyperparameter optimization. Of course, extensive modifications could also be made here, but this would exceed the scope of this work. Finally, the optimized model provides a good generalization on the test data set. In section 6.1.7, the predicted displacements at the next time step are shown graphically. From the similar-sized errors on the training and test data sets, it can be concluded that the model now has a good ability to generalize.

### 6.1.6 Physical loss function

A promising approach, which was presented in [RPK19], is to apply a so-called physical loss function for parameter estimation in neural networks, where the penalty term becomes larger the more the neural network violates known physical properties. This will also be implemented here by approximating the Euler-Bernoulli equation by 4th order finite differences to penalize its violation. Accordingly, the loss function of an MSE-loss is replaced by this physical loss function. It quickly becomes clear that this loss has a high price for the application presented here: For every single grid point in every single sample, the finite difference approximation has to be calculated in order to compute the error. Overall, this increases the training by a factor of 10 due to a great number of data accesses and back-transformations of the data into the actual magnitude, resulting in a long training time. The results of the training with or without the physical loss function are shown in section 6.1.7.

### 6.1.7 Results

In section 6.1.5 the hyperparameters of the neural network have already been optimized, so that now the performance of the model on the test data set can be considered in more detail. After training the deformations shown in figure 6.2 are predicted in the next time step. The predictions on the test data set contain both good predictions with small errors as well as some bad predictions with large errors or unphysical displacements. The root of the MSE-loss (RMSE) averaged over the entire test data set is about 0.02 $m$. The choice of hyperparameters and the number of training epochs has a large impact on the predictions, as demonstrated by the unphysical examples. The more epochs the neural network is trained, the more the deviations between neighboring points disappear, thus the smoother the solution becomes. Similar results are obtained for both the MSE-loss and the physical loss function. This means that no significant advantages can be found here by using the physical loss function compared to the MSE-loss, which is why this loss function will not be considered in more detail in the further process of this work. The many data accesses and transformations are not worth the effort, even if the training time could still be significantly reduced by a more efficient implementation. Of course, this does not mean that the physical loss function is unsuitable for this problem; after all, it yields comparable results to MSE-loss. Overall, this section has shown that it is possible to approximate the solution of the Euler-Bernoulli equation using machine learning techniques, although the results are certainly too inaccurate to fully replace a structural solver with a neural network. This could still be partially modified if, for example, only a fixed test case is considered, but this will in turn lead to a less applicable solver.

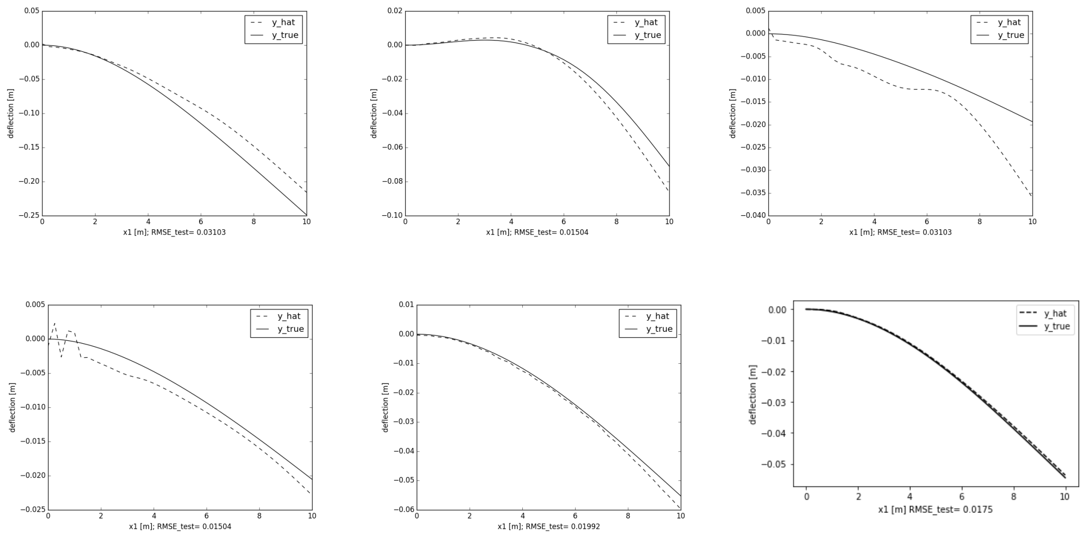


**Figure 6.2:** Predicted ($\hat{y}$) and exact ($y_{true}$) deflections of the 1-dimensional beam for different configurations and time steps in the test data set.

## 6.2 Kármán vortex street

The so-called Kármán vortex street is selected as a test case for learning a fluid flow. This describes the flow of fluid behind an obstacle and is particularly exciting because of vortices arising there in the flow. For this purpose, as presented in [RPK19], a flow is considered here which has a cylindrical obstacle of diameter $1m$ at the origin of the coordinate system and which flows in at the left edge with a uniform velocity profile of velocity $u = 1\frac{m}{s}$. Up to the obstacle, a stationary flow is formed. The flow behind the obstacle is now simulated in the range $[1, 8]\ m \times [-2, 2]\ m$, while the entire numerical computational domain has boundaries of $[-15, 25]\ m \times [-8, 8]\ m$. A kinematic viscosity of $\nu = 0.01$ is chosen so that the flow has a Reynolds number of $Re = \frac{\rho u D}{\nu} = 100$.

### 6.2.1 Navier-stokes equation

The solution of this fluid flow can be calculated numerically using the Navier-Stokes equation, which is derived from Newton's second law, the momentum balance. This is given by

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

for the incompressible fluid flow in two dimensions (2D) considered here.
In this equation, $\boldsymbol{u}$ is the flow velocity, $\nu = \frac{\mu}{\rho}$ the kinematic viscosity with $\mu$, the dynamic viscosity and the density $\rho$. Further, $p$ is the pressure and $\boldsymbol{g}$ represents all body accelerations like gravity. The Navier-Stokes equation is a set of second order partial differential equations, which is derived from the law of conservation of momentum. It will be solved in the following for the fixed test case described above using a neural network.

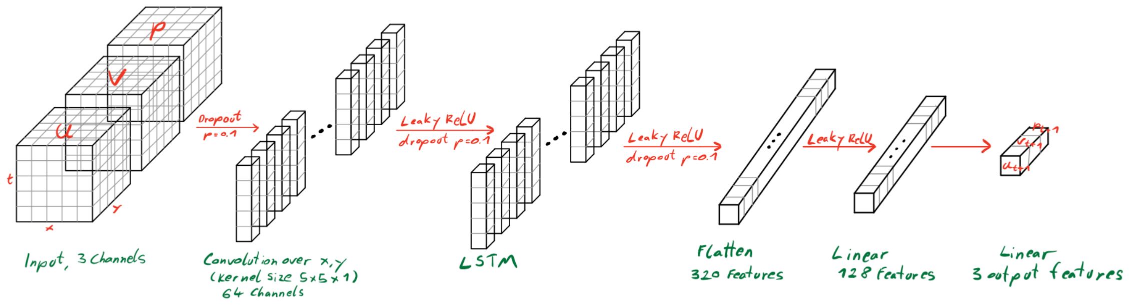


**Figure 6.3:** Overall network architecture for the Kármán vortex street. The input is the local flow field at the previous five time steps and predicted is the flow at the next time step for one discretization point.

## 6.2.2 Modeling

The goal here is to predict the local flow properties in the next time step based on the local flow field (x-velocity, y-velocity, pressure) at a discretization point and its neighbors in the previous time steps. For this purpose, a neural network consisting of a convolutional neural network, a long short-term memory, and a linear regression is implemented. As a reference solution, a numerical solution of this test case provided in [RPK19] is used, which consists of 100 discretization points in x-direction, 50 discretization points in y-direction, and 200 time steps. The overall architecture is shown in Figure 6.3. According to the problem definition, an input tensor of size $(5 \times 5 \times 5 \times 3)$ (x-neighbours, y-neighbors, history, features) is passed to the neural network and analyzed in the convolutional neural network with a kernel size of $(5 \times 5 \times 1)$ for spatial patterns between the data point and its neighbors in each input feature. The convolutional layer consists of 3 input and 64 output channels after hyperparameter optimization. After convolution, the nonlinear activation function leaky ReLU is applied to the data and the hidden features are passed to a two-layer long short-term memory, which uses them to estimate the hidden features in the next time step. After the long short-term memory, the nonlinear activation function leaky ReLU is again applied to the data and finally, a prediction is made for the flow features at the next time step by linear regression from the hidden features. Thus, the output tensor has a size of $(1 \times 3)$ and contains the prediction for the considered discretization point in the next time step based on the given local flow field in the last five time steps.

This approach reduces the total spatial flow field by the outermost two discretization points at each pass through the neural network due to convolution with the spatial neighbors, which results in prediction loss at the boundaries. To fix this, in a preprocessing step, the entire flow field is extended at the edge by two discretization points, so that the discretization of $100 \times 50$ grid points becomes a discretization of $104 \times 54$ grid points, whereby the mesh width remains constant. The padding is performed as shown in Figure 6.4 in such a way that a linear padding respectively a mirroring of the boundary values is used. Thus, the size of the flow field now remains constant, which also allows a further prediction for distant time steps in the future, in that, starting from the given data, the predictions for the next time step are always used to predict the time step after next. In principle, this allows (with sufficiently small errors) the prediction of the flow field up to the desired end time based on the first time steps and a model trained with them.

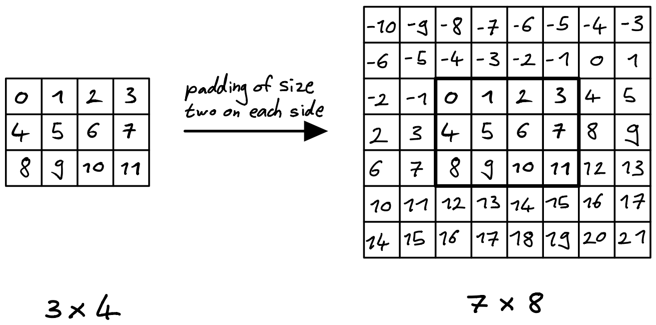


**Figure 6.4:** Padding to keep the domain size constant. Padded values are computed by subtracting the reflected values from two times the edge value.

### 6.2.3 Training

For training, the data set is initially processed in such a way that each sample in the training data set consists of a four-dimensional tensor, which contains the flow features of a discretization point and its two neighbors left, right, top, and bottom at a time step $t_n < t_{maxTraining}$ and additionally the four preceding time steps $t_{n-4}, \ldots, t_{n-1}$. Thus, the local flow field of the last five time steps is passed. In each case $(u, v, p)$ of the discretization point at the time step $t_{n+1}$ is chosen as solution. The neural network shall be trained later using its own predictions to improve the accuracy of the predictions. Therefore, for each training sample, the $useOutputSteps \geq 0$ solutions further in the future are also stored. Accordingly, a test data set is generated, which contains the flow features for $t_n \geq t_{maxTraining}$ at time steps $t_{n-4}$ to $t_n$ and the solution at time step $t_{n+1}$ for error determination, which of course is not passed to the neural network for training.

One percent uncorrelated Gaussian noise is added to all training inputs to obtain more robust results. In addition, all data are linearly transformed to the range of values $[0, 1]$ by min-max scalers for each feature to avoid vanishing or exploding gradients in training. For training the neural network, the Adam optimizer is used with an initial learning rate of $1 \cdot 10^{-3}$. MSE-loss is applied as the loss function to calculate the error gradients. To prevent overfitting, dropout with a dropout probability of $p_{Dropout} = 0.1$ is applied at the beginning, after the convolutional layer, and after the first linear layer. A batch size of 1024 samples is used in the training process and the neural network is trained until overfitting is detected. In a further run, this is then prevented by stopping early and the best possible performance is achieved. During training, the prediction for the next time step (i.e. $t_{n+1}$) is used for the next $useOutputSteps$ to predict the solution in the time step after next (i.e. $t_{n+2}$). This significantly improves the performance of the neural network, as shown in section 6.2.4. The model is finally trained 20 epochs, which takes 18 minutes on an Nvidia GeForce GTX 1050 Ti.

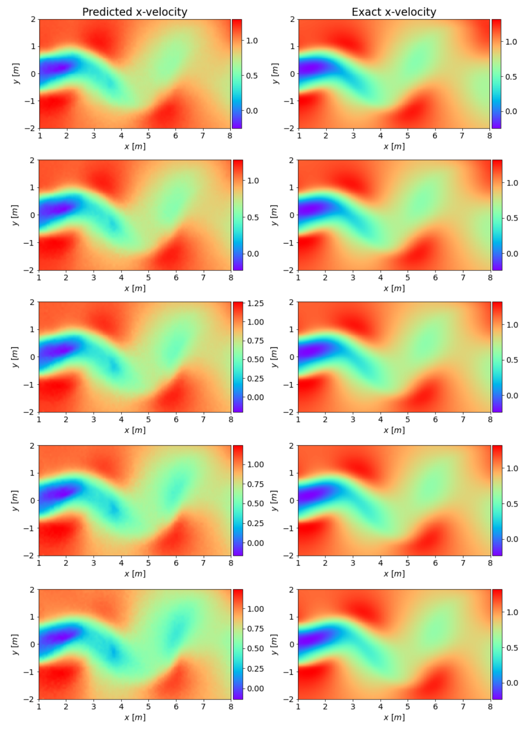
### 6.2.4 Validation

As a matter of principle, simulation results must always be verified for their actual applicability. The most beautiful pictures and the lowest error rates are worthless if the simulation, in the end, does not fulfill the actual requirements that were set at the beginning. In the case of the application of machine learning techniques, this means in particular that the generalization capability of a model is checked on samples that are unknown to the model. Here, this means that after training the neural network, the test data set and appropriate visualizations are used to check how well the fluid flow continues to be predicted. As expected, a rather poor performance on the test data set was found after the first run, which makes it necessary to optimize the hyperparameters of the neural network. Thus, one percent uncorrelated Gaussian noise is added to the training data (after optimization) and dropout is applied with a dropout probability of $p_{Dropout} = 0.1$ at the beginning, after the convolutional layer, and after the long short-term memory layers. Further, the number of hidden neurons in each layer of the neural network is adjusted so that the deviations of the errors between the training and test data sets become smaller. Initially, the neural network is trained only for the next time step, so it always uses the data at time steps $t_{n-4}$ to $t_n$ to predict the flow at time step $t_{n+1}$. In the test run, in accordance with the task, the results are then also considered for time points lying more than one time step in the future. It has to be stated that the results for the first time step are very good, but already at the second, third time step in the future they are no longer convincing and for time steps lying even further in the future they are no longer comprehensible. Therefore, a new method is used to train the neural network. Here the training is modified in such a way that after the backward pass and the updating of the weights of the neural network the data of the time step $t_{n-4}$ are deleted and instead the data of the current prediction of the time step $t_{n+1}$ are attached to the inputs. This is then used to estimate the solution at time step $t_{n+2}$ in a new forward-pass and this is compared to the numerical reference solution to again calculate the loss and gradients of the errors. This is repeated $useOutputSteps$ $(\geq 0)$ times, where $useOutputSteps = 0$ stands for training as before. In this process, it is further ensured that the neural network in training is not passed any data at time steps $t_n > t_{maxTraining}$. If a prediction is no longer possible further into the future, this sample is simply iterated less far into the future. This allows much better results to be obtained on the test data set, as shown in the following section. Furthermore, for this periodic problem, it must also be ensured that the model is only trained and tested on data that lie within one period. Otherwise, there are samples in the training and test data set which are approximately the same or which are contained several times in the training data set and thus no longer provide any additional value. In the data set available here, a periodicity can be determined every 62 time steps, with $n_{training} + (n_{passedSteps} - 1) + n_{pictures} = 12 + (5 - 1) + 9 = 25$ this is thus always ensured.

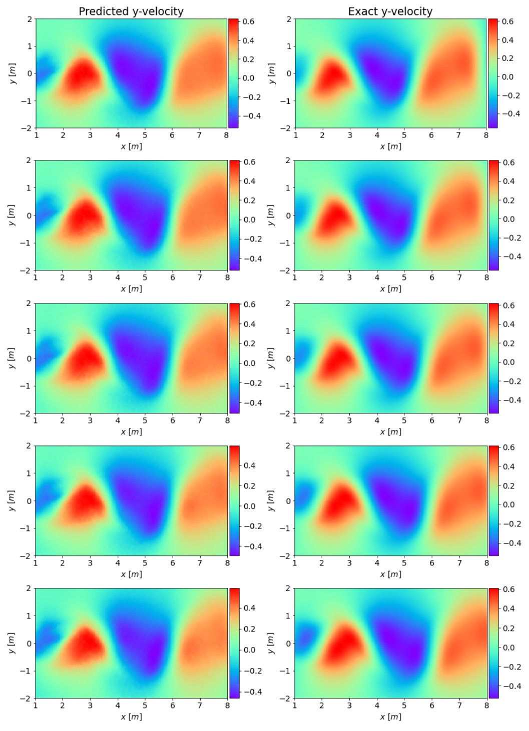
### 6.2.5 Results

After it was verified that the model does not suffer from overfitting and has a good ability for generalization, the results will be presented in the following. For this purpose, it was decided to show both, the predicted and the actual flow in the whole computational domain for the first nine time steps in the test data set. Therefore, the fluid flow in x- and y-direction is shown in Figure 6.5 and the pressure is shown in Figure 6.6.

From these results, it can be seen very well that the neural network has learned the way the fluid flows and what kinds of changes can be expected in the area. Thus, all quantities are predicted in the correct order of magnitude and the deviations at individual points in the considered area are minimal. Inaccuracies, such as the rather unclear boundaries between areas of high and low velocity, propagate only slowly in the prediction. We observe the largest error for pressure prediction. Remarkable is the fact that for the shown nine time steps of prediction only the data of 16 time steps were used for training, i.e. with very small amounts of data already solutions are found, which agree very well with the numerical reference solution.



**(a)** x-velocity



**(b)** y-velocity

**Figure 6.5:** Predicted (left) and exact (right) fluid flow in the Kármán vortex street. From top to bottom: $1^{st}$, $3^{rd}, \ldots,$ $9^{th}$ predicted time step based on the true fluid flow at the last five time steps as an input to the neural network.

Additionally, the differences (errors) between the predicted and the actual, numerically determined, flow fields are particularly interesting for the evaluation of the model in order to be able to quantify errors. The averaged relative errors

$$Error_{Relative}(y,\ \hat{y}) := \frac{1}{N}\sum_{k=1}^{N}\left|\frac{\hat{y}_k - y_k}{y_k}\right|$$

are shown for each time step in figure 6.7. It can be seen that the relative errors propagate with time. The further the predicted time step is away from the training data, the more often arising errors add up and the larger the averaged errors in the whole area become. An exception to this is the y-velocity, for which the errors become smaller until the eighth time step. This shows that the developed solvers give very good results for the first time steps, but spreading errors cannot be compensated anymore. Nevertheless, thanks to the predictions reused in training, the solution is stable for several time steps with desirable quality.

In order to visualize the local causes of arising errors and their propagation, the relative errors obtained in the test data set for each discretization point in the computational domain are presented below. Here, the errors in the flow velocity in the x-direction, y-direction, and the pressure are shown in Figure 6.8.

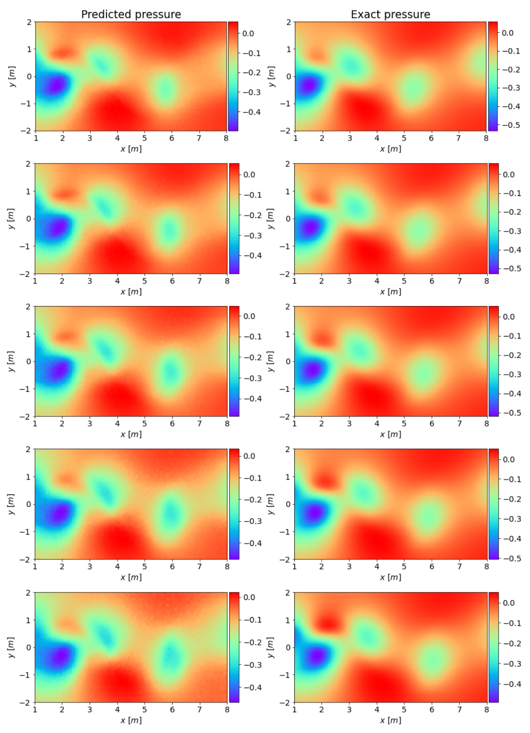


**Figure 6.6:** Predicted (left) and exact (right) pressure in the Kármán vortex street. From top to bottom: 1st, 3rd, . . . , 9th predicted time step based on the true fluid flow at the last five time steps as an input to the neural network.

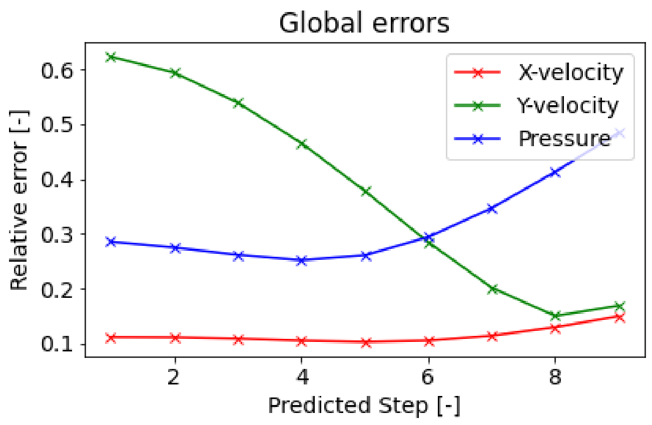
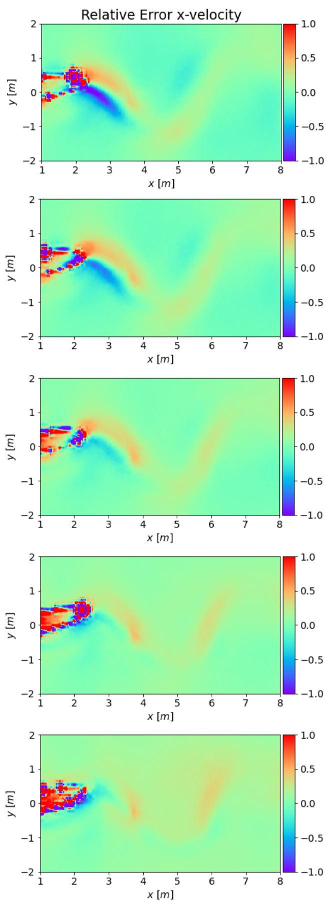
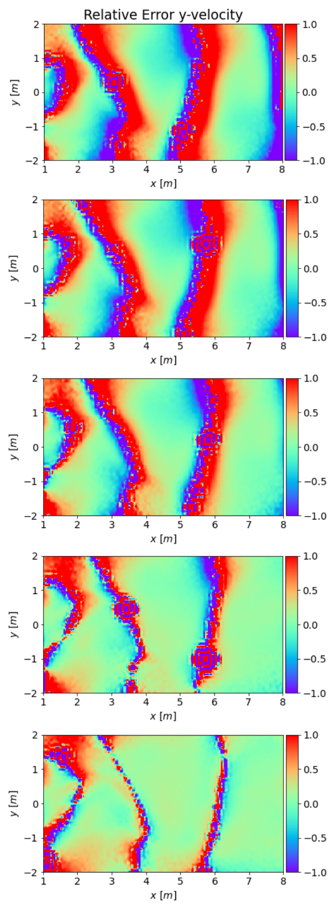


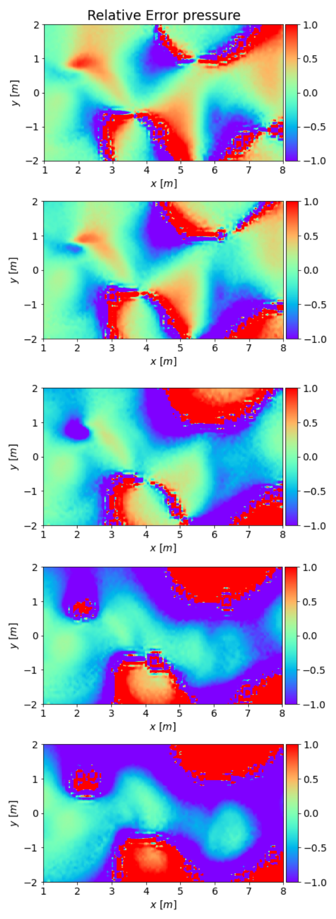
**Figure 6.7:** Global prediction errors for the Kármán vortex street and increasing number of predicted steps without new information from the numerical solver.



**(a)** x-velocity



**(b)** y-velocity



**(c)** pressure

**Figure 6.8:** Local relative errors in the predicted fluid flow in the Kármán vortex street. From top to bottom: $1^{st}$, $3^{rd}, \ldots,$ $9^{th}$ predicted time step based on the true fluid flow at the last five time steps as an input to the neural network.

In the x-velocity, the largest errors occur where the solution changes most rapidly. Both too high and too low velocities have been predicted so that a lack of capacity of the model cannot generally be concluded here. For the y-velocity, errors occur especially where transitions between upward and downward flow occur. It looks as if the neural network predicts too large a change at the beginning, which is only actually achieved after about six time steps. At the same time, the changes in the following steps are very small, so that the predicted solution and the reference solution become more and more similar as time progresses, until after the eighth time step the errors become larger again. The propagation of small errors across the domain can be observed particularly well in pressure, where, starting from inaccuracies in the first predicted time step, larger and larger contiguous regions predict either a too high or too low pressure, until after about seven time steps the solution has developed contiguous error regions and the solution is no longer accurate there.

In the following, the solvers that will later be coupled together to perform the simulation of the fluid-structure interaction problem will be developed. The coupling library required for this is therefore now briefly considered in more detail.

# 7 Coupling with preCICE

It was already discussed in the theoretical part which opportunities exist to couple equation solvers with each other, so that also coupled systems of equations can be solved numerically. In principle, this coupling works the same way for neural networks to simulate fluid-structure interaction as it does for numerical solution methods: Either a common, large neural network is written that simultaneously determines the solutions of both domains, or the two neural networks for solving the single-physics equations are exploited as they are mutually coupled in such a way that they exchange solution at the common boundary to predict the solution at their own domain. The first concept requires a neural network that is trained with the information of both domains. The overall task to be solved changes both in the way data is processed and in the form of the outputs. This means neural networks already trained on single-physics cannot continue to be used and must be retrained. In particular, this also requires the hyperparameters to be re-optimized. Thus, this can increase the complexity and cost of the solution. However, by using a coupling library, we are able to couple two single-physics neural network for multi-physics simulation. We use preCICE [BLG+16; HSG+10] coupling library for this purpose, which is schematically shown in Figure 7.1.

The preCICE library provides data communication between solvers, boundary data coupling, and equation coupling for strongly coupled simulations. A .xml file is used to define each multiphysics simulation and to set convergence conditions. Afterward, the already developed neural networks are each integrated into a code framework needed for preCICE, which sets up the communication. This approach is relatively fast to implement and at the same time allows for a variety of changes in the coupling. The actual coupled simulation is then started and controlled by preCICE. In this work, both an explicit and an implicit coupling scheme are used and the predicted solution is passed to the other neural network at each time and iteration step, respectively, to solve the coupled systems of equations.

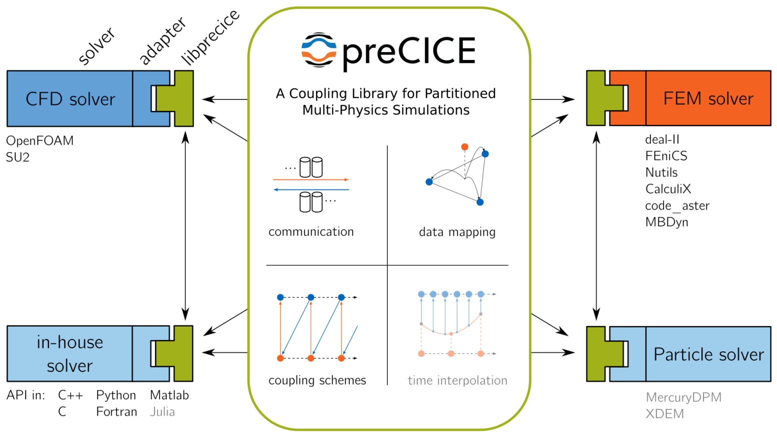


**Figure 7.1:** Schematic overview of the coupling library preCICE [BLG+16].

# 8 Multi-physics neural network solver

All experiences and sub-projects gained so far shall be combined in this last content chapter. Here, single-physics neural network solvers are specifically developed to predict the solution of a coupled fluid-structure interaction problem in the next time step in their own subdomain using the solutions or predictions of the last time steps in both subdomains. As a test case, a one-dimensional elastic tube is presented here as an example of a multi-physics simulation steered by preCICE coupling library [Tut21]. In this test case, the flow through a flexible tube is simulated as shown in Figure 8.1.

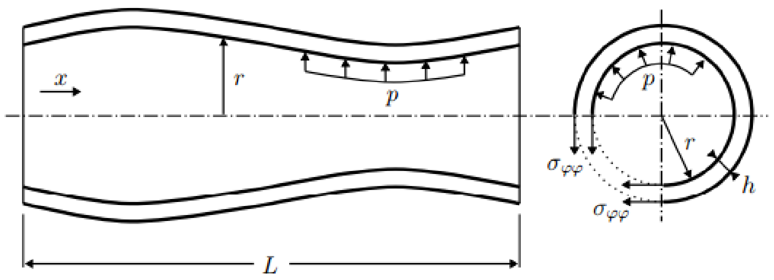


**Figure 8.1:** 1d elastic tube schematic geometry and parameters [Gat14].

For the distributed approach to solving the coupled equations, first, the neural networks are implemented in section 8.1 and 8.2, which each predict the solutions in their own subdomain and then these are coupled together with preCICE in section 8.3.

## 8.1 Single-physics solver for fluid flow

The equations needed for solving the fluid flow in the elastic tube are conservation of mass and momentum and are derived in [DBHV08]. These can be written as

$$\frac{\partial a}{\partial t} + \frac{\partial (av)}{\partial x} = 0,$$

$$\frac{\partial (av)}{\partial t} + \frac{\partial (av^2)}{\partial x} + \frac{1}{\rho}\left(\frac{\partial (ap)}{\partial x} - p\frac{\partial a}{\partial x}\right) = 0.$$

Here, an incompressible fluid of density $\rho = 1000\ \frac{kg}{m^3}$ flows from the left with inlet velocity $v_{in} = 10 + 3\sin(10\pi t)\ \frac{m}{s}$ into an elastic tube of length $L = 10\ m$, initial cross-sectional diameter $1\ m$ and initial pressure on the wall of $p = 0\ Pa$. Gravity is neglected and the axial symmetry of the tube is exploited in the derivation to reduce the system to one dimension. The reference solution

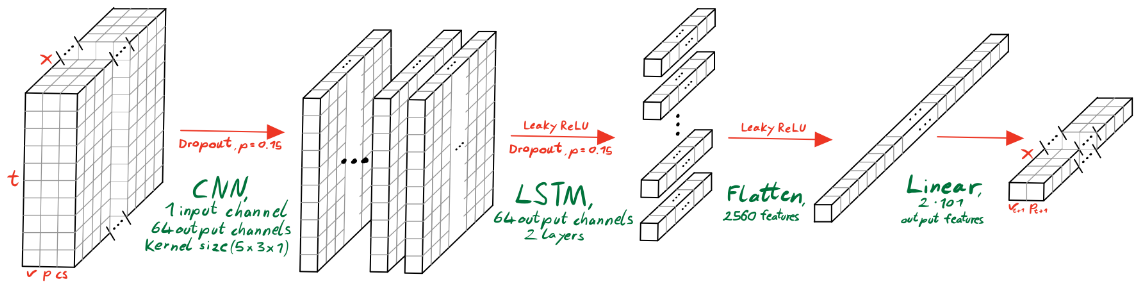


**Figure 8.2:** Schematic view of the neural network fluid solver.

obtained by classical numerical solution methods discretizes the domain into $nDiscretization = 101$ equidistantly distributed grid points and time step sizes $\tau = 0.004\ s$. These parameters are also chosen for solving the equations using neural networks. The data set used for training the neural networks is created using the numerical solution of the coupled equations. It contains the velocity in the x-direction $v(x)$, the pressure $p(x)$ and the cross-sectional diameter $cs(x)$ at each time $t_n = n\tau$ with $n \in \mathbb{N}_0$ up to the arbitrary end time $t_{max}$.

The goal of the single-physics neural network for fluid flow prediction is to use the solutions of the previous time steps from both subdomains to predict the solution of the fluid equations, i.e., the pressure $p(x)$ at the next time step. At the same time, for an implicit coupling scheme, this should allow the prediction of the solver of the structural equations in the current iteration to be used as a boundary condition for the fluid flow in the next iteration, so that even for strongly coupled problems the solution can be predicted iteratively.

### 8.1.1 Modeling

In order to model this, a neural network is implemented as in chapter 6 consisting of a convolutional neural network, a long short-term memory neural network and linear regression layer, as shown in Figure 8.2. The training data set from the numerically computed reference solution is processed such that the first $N_{train} = 32$ possible time steps are used for training. These contain for each time step to be predicted the history of the last $passedSteps = 10$ preceding time steps. A training sample here contains the data for the entire elastic tube, so it is stored as a tensor of size $(passedSteps \times nDiscretization \times 3)$. Therefore, to extract local information from it, a kernel of size $(1 \times givenNeighbor \times 3)$ is applied in the convolutional neural network. This, thus, applies a convolution for every single time step in history over the locally given $givenNeighbour = 5$ and all features of the flow (velocity, pressure, cross-sectional diameter), so that thereafter the local spatially learned patterns in the data at each of the $passedSteps$ time points prior to the new prediction are present as hidden features. This corresponds to a tensor of size $(passedSteps \times 97 \times 1)$. For convolution, 64 different kernels are used, which then produces 64 such tensors. After each layer, the nonlinear activation function leaky ReLU is applied to the hidden features. The spatially local patterns of the convolutional neural network are passed to a two-layer long short-term memory neural network, which extrapolates these features from the past into the future. After applying the nonlinear activation function leaky ReLU, the data is flattened and written into a vector of length 2560, on which finally performing a linear regression forms the prediction of velocity and pressure at the next time point. This prediction does not get any further activation function respectively uses the identity function, so that the predictions can take continuous, in particular also negative, values in $\mathbb{R}$. This is physically justified, since both the pressure and the velocity can take negative and positive values, depending on the setup of the tube. The output tensor has a size of $(2 \times nDiscretization)$ corresponding to the input tensor since the flow velocity and pressure are to be predicted for each grid point.

### 8.1.2 Training

To train the neural network, the training data set is used, which contains the history of the past time steps and as a target the flow features in the next time step for each sample. Only the first possible time steps will be used in the training process, so the neural network must already learn the physics behind the data from the settling process. Thus, this learning strategy can be extended quite generally to other physics problems, for example, by having classical numerical solvers determine the systems of equations in the first time steps and at the same time, the neural network learns with these results and then takes over the function of the solver for later time steps.

To create the data set for training and testing the neural network, the classical numerical solver of this test case was modified to store the output data for each time step in a .npy file. These are read in at the beginning of the training and preprocessed accordingly. One percent uncorrelated Gaussian noise is applied to the input data and each feature is transformed to the range of values $[0, 1]$ by a min-max scaler. This is to avoid vanishing and exploding gradients in the training. The Adam optimizer, which has already shown good performance in the single-physics test cases, is used for training. The initial learning rate is defined as $lr = 1e - 03$ and the loss function is the mean relative squared error

$$L^{MRelSE}(y,\ \hat{y}) := \frac{1}{N} \sum_{k=1}^{N} \frac{(\hat{y}_k - y_k)^2}{y_k^2} \tag{8.1}$$

used. Again, dropout is shown to be a good regularization method with a dropout probability of $p_{dropout} = 0.15$. This is applied to the input data, after the convolutional neural network, and after the long short-term memory neural network. Since only a comparatively small amount of data is stored in the test data set, this allows the forward pass to be performed all at once on the GPU.
The neural network, in order to determine how many epochs need to be trained for the best result, is trained until overfitting can be detected and then reset to a state where generalization ability is at its best. This can be easily implemented by using a separate validation data set and determining the training and validation error of the current model after a fixed number of epochs, in this case 100. To do this, the model is set in an evaluation state so that no information spills over from the validation dataset into the training in this process. Afterward, the current model is saved and further trained. Thus, it is always possible to revert to a previous state and the model does not have to be trained several times to actually obtain the best possible performance. Figure 8.3 shows an error plot created after hyperparameter optimization in the validation process, here the best performance on the test data set is achieved at about 400 epochs and the corresponding model is used in the actual application. At the same time, a discrepancy between training and test errors can also be observed for further training. The training errors become smaller than the test errors, which is a clear indicator for overfitting. The model needs only 10 seconds to train the 400 epochs on an Nvidia GeForce GTX 1050 Ti.

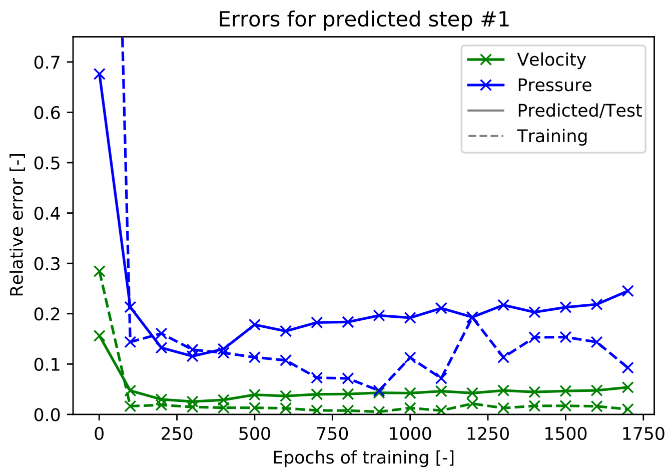


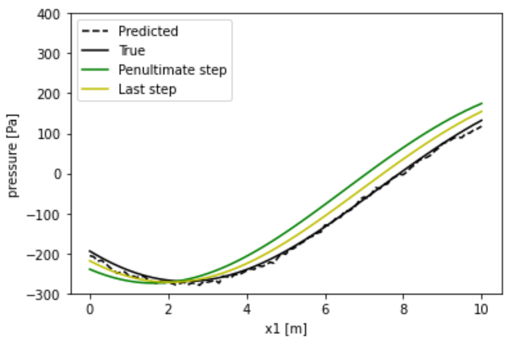
**Figure 8.3:** Comparison between errors for the fluid neural network. Overfitting can be seen if the network is trained to many epochs.

### 8.1.3 Validation

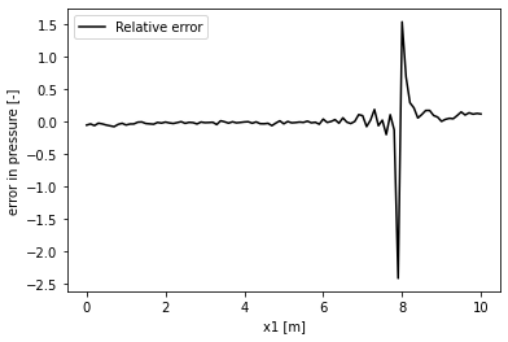
Figure 8.3 shows relative errors of about 13 percent in predicting pressure at the next time step and about three percent errors in speed on the test data set at best performance. To achieve this, regularization techniques must be applied in the training process. Thus, different magnitudes of uncorrelated Gaussian noise are applied to the training data and the corresponding best performance within 1000 training epochs is determined. This shows that the predictions become best when only 0.05 percent noise is added to the data.
Furthermore, dropout is applied during the training process. To find out how large the dropout probability has to be for effective training, different dropout probabilities are used for training, the neural network is trained again for 1000 epochs each with otherwise constant configuration, and the training errors obtained afterward are compared with the test errors. This shows that a dropout probability of 15 percent is the most effective.
Experiments showed that $passedSteps$ must be chosen as a function of $\tau$, so that at least the same amount of time is always passed as history. Thus trained, the optimized model provides good generalization on the test data set.

### 8.1.4 Results

Figure 8.4 shows exemplary a prediction of the neural network against the numerically determined reference solution for a sample in the test data set. It can be seen that the deviations between prediction and target are in the range of a few percent and fluctuate randomly around zero as long as the solution itself is non-zero. In this case, although the relative error is very large, in absolute terms the prediction there is as good as anywhere else. This noise of errors around zero shows that the neural network has no systematic modeling errors but has learned as well as possible what the physics behind the input data is. Further analysis reveals that for the other time steps, a comparable prediction quality can be seen, with one exception, which is shown in Figure 8.5 for $t = 0.224\ s$: when the neural network is asked to predict a time step, which according to the numerical solution



(a) Predicted and true pressure. Coloured are shown the true pressure distributions in the last two time steps before this prediction.



(b) Relative error between true and predicted solution.

**Figure 8.4:** Pressure distribution in the elastic tube for the time step $t = 0.172s$.

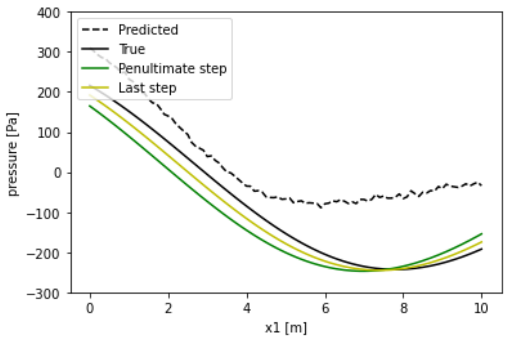


**Figure 8.5:** Large error for completely unknown example. At the right end, a lower amplitude of the pressure is predicted than it is in reality. This shortens the predicted period duration.

physically has a very large negative pressure distribution at the right end, this amplitude is predicted much lower. A search for the cause of these large errors makes the following clear: At the beginning of the simulation ($t = 0\ s$) the elastic tube is not filled with fluid and consequently undeformed. At this time, the fluid begins to flow into the tube, increasing the pressure at the initial left part and stretching the tube. At the same time, the right end is still undeformed. In the process of training, the first wave flows exactly to the right end of the tube, the last sample provided in the training is at time $t = 0.164\ s$. Thus, in the whole training process, only the settling process is provided to learn from. If the simulation is subsequently continued on the test data set, the neural network continues to predict an oscillation, but at a slightly higher frequency and with much less negative pressure amplitude at the right end than in the numerical solution. This shows so far a limit of the generalization ability of the trained neural network and should be investigated in more detail in further studies on this topic. A new approach would be to use as input not the global flow field but many local neighborhoods, as presented in 6.2. Since the physics at each individual point is independent of distant points, this new approach would preclude the neural network from learning connections here that are unphysical. However, this research is beyond the scope of this thesis. Overall, the neural network shows good generalization ability, in particular, the predicted solution remains stable and similar to the actual solution over the entire simulation period. At the same time, however, it has learned a slightly higher frequency in the data than that of the reference solution.

Thus, it cannot be excluded that the network has again allowed global dependencies, which are not physically justifiable, from the local patterns created by the convolution with small kernel sizes in the linear regression layer. A local approach could solve this problem, but for this, the data sets to be processed increase due to many input tensors shifted by one discretization point each and at the edges, possibly by padding, ways have to be found to prevent a reduction of the output domain.

## 8.2 Single-physics solver for structural deformation

The structure equations for the tube walls are given by a linear elastic constitutive relation law with the scalar circumferential stress [DBHV08]

$$\sigma_{\phi\phi} = E\frac{(r - r_0)}{\partial r_0} + \sigma_0.$$

Here, $E = 10000\ kPa$ is the elastic modulus of the tube wall, $r$ the deformed and $r_0 = 1\ m$ the initial cross-sectional diameter and $\sigma_0$ the circumferential stress at reference position $r_0$. The discretization remains the same as for the fluid flow.

Just as shown in the previous section, a single-physics neural network is also implemented for the structural deformations. Here, the goal is to use the solutions of the last time steps from both subdomains to predict the solution of the structural equations, i.e., the deformation and cross-sectional diameter $cs(x)$, respectively, at the next time step.

At the same time, if an implicit coupling scheme is used to couple the two neural networks, it should be possible to use the prediction of the solver of the fluid equations in the current iteration as a boundary condition for the structural deformation in the next iteration, so that even for strongly coupled problems the solution can be predicted iteratively.

### 8.2.1 Modeling

For modeling these deformations, the neural network architecture of the single-physics fluid flow solver is modified to predict appropriate outputs. It again consists as presented in Figure 8.6 of a convolutional neural network to extract local patterns in the data at the beginning, a long short-term memory neural network to extrapolate the patterns to the next time step, and a linear regression that makes the actual prediction. The training data set from the numerically computed reference solution is processed so that the first $N_{train} = 32$ possible time steps are used for training. These contain the history of the last $passedSteps = 10$ preceding time steps for each time step to be predicted, thus providing the same initial conditions as the single physics fluid solver. This is for compatibility in a real deployment, where there are the same number of time steps as a reference solution for each sub-domain.
A training sample contains the global data for the entire elastic tube and is stored at 101 discretization points as a tensor of size $(10 \times 101 \times 3)$. To detect local patterns at each time step, hidden features are applied with a kernel of size $(1 \times 5 \times 3)$. In this configuration, correlations between velocity, pressure, and cross-sectional diameter are also detected. 64 different output channels are specified, which allows training the kernels on as many different patterns. Again, the nonlinear activation function leaky ReLU is applied to the hidden features after each layer. A two-layer long short-term

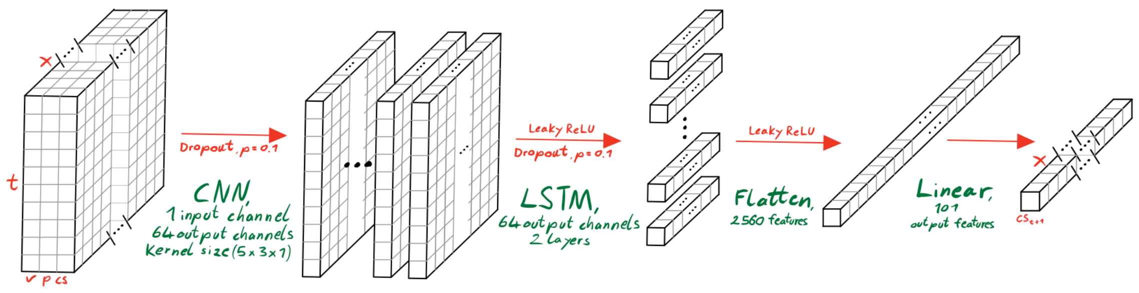


**Figure 8.6:** Network architecture for the neural network structural solver. The only difference from the fluid solver in the architecture is the shape of the output.

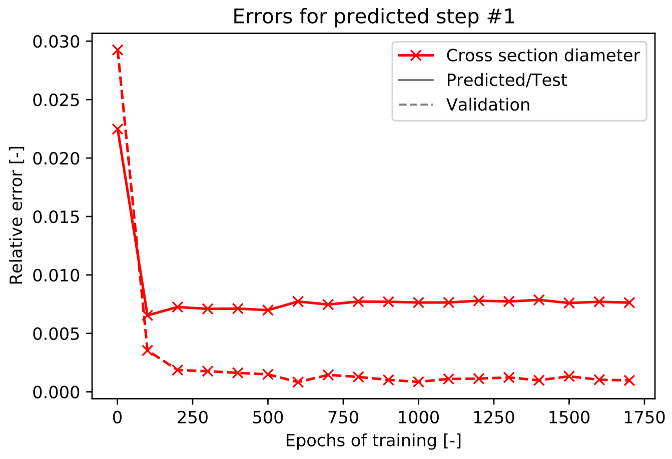
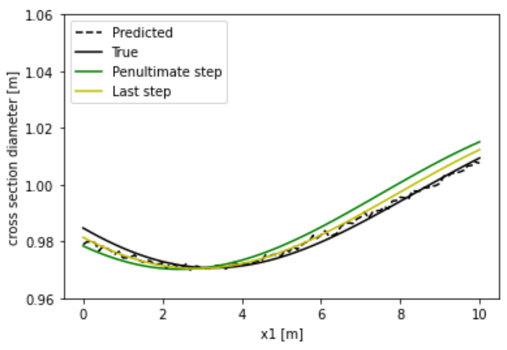


**Figure 8.7:** Validation and test errors for the structural subdomain neural network depending on the epochs of training.

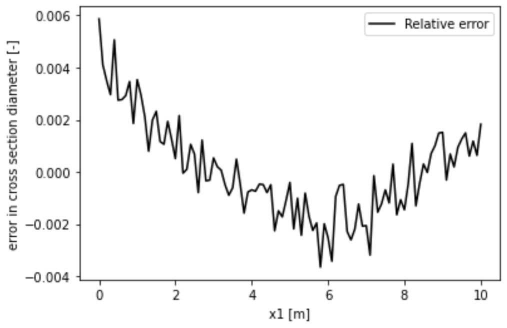
memory neural network is tasked with predicting the future using these features from the past. The output tensor of size $(10 \times 256)$ is flattened and a linear regression is performed at the end, which performs the actual prediction of the cross-sectional diameter $cs(x)$. This no longer gets an activation function, since the final predictions have already been made and written to the output vector of length 101.

### 8.2.2 Training

To train the desired neural network for the structure domain, the data sets required for this purpose must be read in and preprocessed so that they have the properties described in the modeling subsection. The .npy files created by the numerical equation solver are used and processed for this purpose. One percent uncorrelated Gaussian noise is added to the inputs of the training dataset $X_{train}$, and min-max scalers transform each feature into a range of values between zero and one.
For optimization, the Adam optimizer is used, which is assigned an initial learning rate of $lr = 1e - 03$ and the loss function to determine the steepest error descent is mean relative squared error $L^{MRelSE}$ (8.1). Dropout with a dropout probability of $p_{Dropout} = 0.1$ before and after the convolutional neural network and after the long short-term memory neural network thereby



**(a)** Predicted and true cross-sectional diameter. Coloured are shown the true pressure distributions in the last two time steps before this prediction.



**(b)** Relative error between true and predicted solution.

**Figure 8.8:** Cross-sectional diameter in the elastic tube at time step $t = 0.18s$.

improves the generalization ability of the model. Training runs until the best generalization ability is found and the validation loss doesn't decrease anymore. The training and testing errors for the cross-section diameter are shown in Figure 8.7. Here very low relative errors of 0.7 percent are achieved on the training data set, after 500 epochs the quality is best. The model does not suffer from overfitting even for a longer training, but it does not gain any more generalization ability either. The model needs only 44 seconds to train 900 epochs on an Nvidia GeForce GTX 1050 Ti.

### 8.2.3 Validation

Figure 8.7 shows relative errors of about 0.07 percent in the prediction of the cross-section diameter at the next time step on the test data set at best performance. The regularization methods here effectively prevent overfitting to the training data. To determine the hyperparameters needed for this purpose, various magnitudes of uncorrelated Gaussian noise are added to the training data and best performance within 1000 training epochs is determined. A 0.1 percent added noise gives the best predictions. Dropout is applied during the training process. To find out how large the dropout probability has to be for effective training, different dropout probabilities are used for the training, the neural network is trained again for 1000 epochs each with an otherwise constant configuration and the training errors obtained afterward are compared with the test errors. This shows that a dropout probability of 15 percent is the most effective.

### 8.2.4 Results

Like the modeling and training, the predictions of the structural solver are similar to those of the fluid solver. An example from the test data set is shown in Figure 8.8. Here, the cross-section diameter predicted by the neural network and that of the reference solution are compared and the relative errors between them at each discretization point are shown. The predictions are very good, but not perfect. For example, especially at the left beginning of the tube, the diameter predicted is too small and, accordingly, the relative error is highest at 0.6 percent. Since the modeling of the

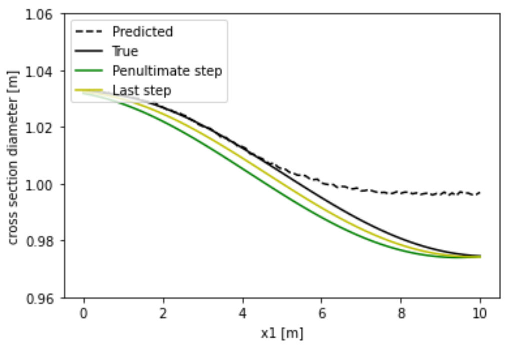


**Figure 8.9:** Sample in the test dataset where the cross-sectional diameter is, in accordance with the fluid solver, not correctly predicted at the right end.

structure sub-domain solver is almost identical to the fluid sub-domain solver, and pressure on the inner wall of the tube leads to deformation due to a linear elastic constitutive law, the problems encountered in the structure solver are also the same as those in the neural network for the fluid domain. At the right end of the tube, the compression that occurs is almost not predicted at all as shown in Figure 8.9 for the time step $t = 0.248\ s$.

Overall, the neural network shows a good usable generalization ability, in particular, the solution predicted for the structure sub-domain remains stable and similar to the actual solution over the whole simulation period. Besides, deviations are shown in samples where compressions occur at the right end of the tube. However, for the goal of coupling neural networks to be able to describe interdependent systems, these deviations are not primarily relevant but can serve as a basis for future improvements if further coupled fluid-structure interaction problems are to be solved by neural networks. At this point, the two sub-domain solvers for fluid and structure can be coupled using the coupling library preCICE.

## 8.3 Coupled simulation with fluid-structure interaction

To solve the coupled fluid-structure interaction problem depending on both sub-domains, the trained neural networks are coupled with each other. The coupling should be able to be executed implicitly as well as explicitly, so that the derived approaches can be flexibly developed further. However, one restriction is justified here: For the implementation, it has to be assumed that the time step size does not change. The neural network gets only the solutions of the last time steps, so far no further variables like the time step sizes, other known variables or parameters of the system. Thus, it is implicitly assumed in the training that the time step sizes do not change, which is therefore also assumed in the application.
The coupled simulation should now predict the partial solution for each sub-domain in a partitioned manner and pass it to the other neural network, which thus iteratively predicts the solution in its own domain that depends on the received information. At the same time, if large systems are to be simulated, the computation can be parallelized. A local approach in the single-physics solvers would even extend this possibility by allowing individual discretization points to be computed on different cores. In the following, the coupled solver will be discussed in more detail.

**Algorithm 8.1** Solving one time step with the neural network solvers.

```
while not converged and it < max it do
    velocity, pressure ← RUN FLUID SOLVER(velocity, pressure, crossSection, t)
    WRITE DATA(velocity, pressure)
    fluidNN.precice.ADVANCE
    crossSection ← RUN STRUCTURE SOLVER(velocity, pressure, crossSection, t)
    WRITE DATA(crosssection)
    structureNN.precice.ADVANCE
    fluidNN.READ DATA(crosssection)
    structureNN.READ DATA(velocity, pressure)
    converged ← precice. ISACTIONREQUIRED
    if converged or it = max it-1 then
        $t \leftarrow t + dt$
        FINALIZE TIMESTEP
    end if
    it ← it + 1
end while
```

### 8.3.1 Structure of the solver

A classical numerical solver for the fluid-structure interaction test case of the elastic tube is provided in [Tut21]. The mathematically derived solvers implemented there are replaced by the neural networks derived in section 8.1 and 8.2. Here, the solution variables of the numerical solvers are the velocity $v(x)$, the pressure $p(x)$ on the wall for the fluid solver, and the cross-sectional diameter $cs(x)$ for the structure solver. That is why these variables were also chosen as target values for the neural networks. In analogy to the methods to be replaced, the solution of a time step proceeds according to algorithm 8.1.

The sub-domain solvers predict the solution at the next time step or return the numerically determined reference solution from the training data set, respectively, if not enough past time steps have been stored at the beginning of the simulation to make a prediction with it. From the time when $passedSteps$ have been stored, the neural network starts to make its own predictions. For later time steps the whole input tensor is rolled through by one time step and the passed data is inserted behind. Thus, the past can be used for the prediction of the next time step without having to pass it explicitly with each call. If an implicit coupling step is performed, the neural network recognizes from the unchanged simulation time $t$ that it must perform an implicit step and performs an update of the inputs to the latest state instead of rolling through all time steps. After that, a new prediction is made as well.
The interface provided by preCICE initializes the neural networks and communication. After the iteration steps of the two solvers, the new solutions are communicated and the interface checks if the solution has converged according to the condition in the .xml configuration file. If another iteration is required, a new iteration step is executed. If the solution has converged or the termination criterion is met, the time step is finalized. Output files such as plots of the solutions or VTK files are also written during this process. If the simulation is terminated, the interface closes the communication channels and releases used memory.

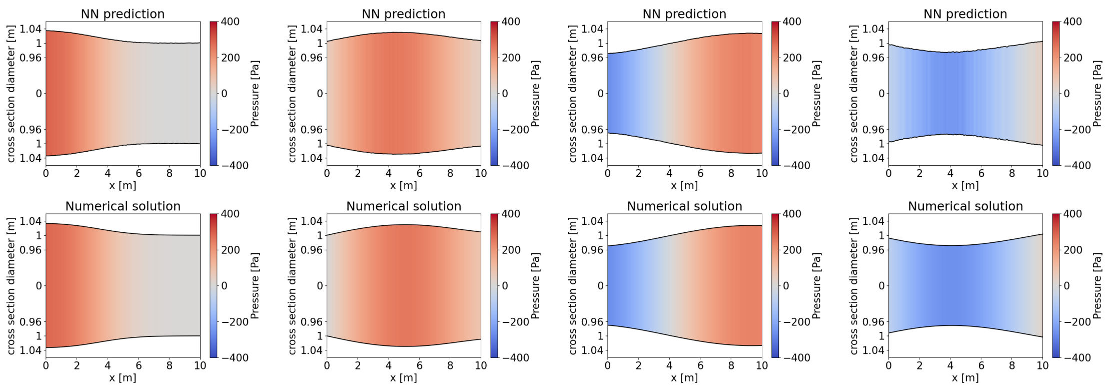


**Figure 8.10:** Predictions of the neural network (top) and numerical solution (bottom) at time steps $t = 0.052,\ 0.100,\ 0.140,\ 0.192\ s$ (left to right) using an explicit coupling scheme.

### 8.3.2 Results

The preCICE interface allows both implicit and explicit couplings of the solvers involved. First, an explicit coupling procedure is applied and thus the fluid flow in the elastic tube is further predicted from the point where the training data set stops. In this process, the fluid subdomain solver always starts first by predicting the pressure and velocity at the next time step, and with this new information, the cross-sectional diameter is predicted in the structural subdomain solver. Both partial solutions together form the solution to the coupled fluid-structure interaction problem and are written to the input of the subsequent time step. Thus, the solution can be computed in all subsequent time steps. For the explicit coupling scheme, the result is the solution shown in Figure 8.10, where pressure and cross-sectional diameter are shown together. Here, the cross-sectional deformations are greatly exaggerated for better visibility. For a comparison with the numerical solution, the latter is shown below the corresponding plot of the prediction.

A very good agreement between the results of the numerical simulation and the predictions of the coupled neural networks is shown. The prediction remains stable until the end of the simulation period at $t = 1\ s$ and maintains its quality. At the same time, however, the same challenges as for the uncoupled solvers become apparent: The frequency of oscillations is about eight percent higher in the neural network prediction than in the reference numerical solution, and locally, fluctuations in the predictions around the actual solution are evident. These results are very much in line with expectations for the solver using coupled neural networks and show that this partitioned approach has the potential to solve more problem areas and can be improved.

One such improvement would be the extension to an implicit coupling scheme already mentioned above. This would allow solutions to become smoother and, in particular, to solve strongly coupled problems with strong mutual interactions in such a way that by updating the boundary conditions, the mutual interactions can be iteratively incorporated more accurately into the solution.
To make this possible, a way must be found to train the neural networks such that a converging solution is also iteratively predicted for a constant time step $t_n$ under updated constraints.
The faster to implement approach for this uses as boundary values the data at the end of the passed history. This is initially the data corresponding to the last time step before the one to be predicted. The currently iteratively predicted boundary values are inserted at this point and a prediction for the

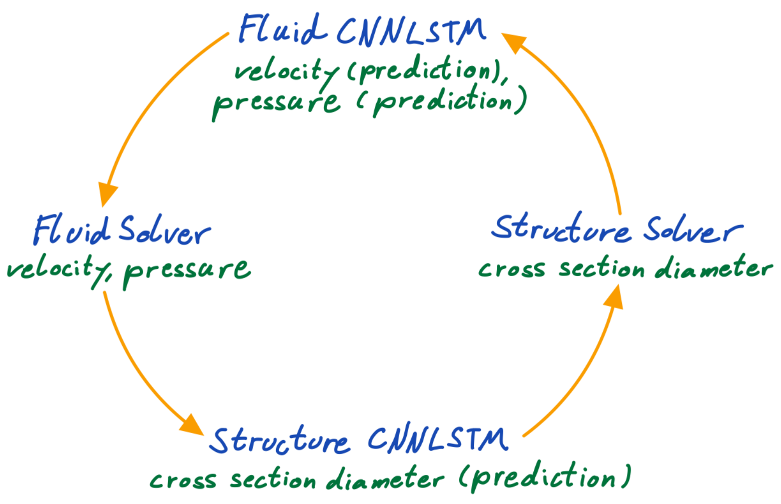


**Figure 8.11:** Sketch of the proposed hybrid solver that predicts a starting solution with the neural networks and improves it with classical numerical solvers.

next iteration is determined. Since the neural networks are trained only with the converged solutions from the training data set, they are not trained again after this change, since the task remains basically unchanged. The coupling scheme is changed to an implicit scheme, with a maximum of 50 iteration steps and a termination criterion of a relative change smaller than $1e-5$ in each solution variable. For this purpose, a quasi-Newton method is used to accelerate the coupling convergence. The results obtained quickly reveal that this approach rarely leads to convergence, and the relative changes between two iteration steps lead to larger, fluctuating changes in the estimated solution due to changes in the input data. In each case, the relative changes are approximately $5e-3$, showing that even small changes in the input can lead to relatively large changes in the prediction. On the other hand, these (too) large fluctuations are not surprising in this approach, since after the first iteration the input tensor changes such that always the last $passedSteps - 1$ time steps remain the same and for the last input is twice the other time steps. At the same time, the information from this time step is lost. To prevent this from continuing, the architecture is changed so that the input contains $passedSteps + 1$ time steps. The first $passedSteps$ remain unchanged in the input and at the end, the flow field at the current time $t_n$ is written again initially. After each iteration of the solver, this is overwritten by the current predictions. The neural networks are also trained appropriately for this purpose. As required in the application phase, the solution at the last possible time step $t_n$ is initially set as the starting estimate for $t_{n+1}$ and then the forward and backward passes are alternately performed for the fluid and structure neural networks, the current predictions are written to the input and iterated on. Thus, if more than one iteration is specified, an investigation of coupled training may still occur.

In conclusion, these investigations indicate that extensions of machine learning to coupled fluid-structure interaction problems must go beyond the scope of this thesis. The potential application areas are large, and at the same time, challenges have been highlighted that require creative new solution methods. For example, the relative changes for both applied methods do not converge to zero but level off at about $1.5e-4$. Contrary to expectations, the implicit coupling scheme was found to produce lower smoothness than the explicit coupling scheme. For the first approach, this can be explained by the changes in inputs with which the neural networks were not trained. For the second approach, different inputs should give the same prediction. At the same time, the data at the end of the input tensor changes at each iteration, providing other hidden features that are passed to the long short-term memory neural network. This, in turn, does not seem to learn in the process that these constraints are meant to help the solution converge.

To sum up, therefore, the extension to implicit coupling has revealed many pitfalls, the causes and fixes of which should be investigated in further work on this topic. At the same time, a stable solution could be achieved for both the implicit and explicit coupling schemes, which could be found without classical mathematical modeling and at the same time with relatively low computational effort. This clearly shows the potential of machine learning of fluid-structure interaction to be unfolded. It is completely clear that at this point we are still at the beginning of the development of such methods and the solution methods presented here may be far from mature. Furthermore, the goal should remain realistic and for this purpose, in an investigation following this work, the hybrid models presented in the literature can be analyzed, where here specifically as sketched in Figure 8.11, the numerical solvers with the neural networks in cooperation predict an initial estimate and this is then iterated to convergence in fewer time steps to the physical solution.

# 9 Outlook

The work on this topic was a first step towards the project of extending machine learning methods to simulation of fluid-structure interaction. Many smaller subprojects have helped to identify opportunities and challenges and to seek approaches to address the challenges. After all, it is the things that do not work as expected that often open up great opportunities to discover new things and create approaches that get closer to the reality behind artificial intelligence. We would very much like to look at this topic area generally, realize even more approaches and find solutions that are worthy of the scientific claim behind this work. On the other hand, clear boundaries have to be drawn where to investigate further and which questions are perhaps not important enough to be answered in detail. Thus, at some points, reference could only be made to further research, as the questions raised could not be answered quickly and clearly but even raised more questions.

Further research, highly interesting from the perspective of this thesis, can take up here and investigate the following aspects:

Can local approaches, where the inputs contain only a neighborhood of the point under consideration, better learn the physics behind the data? Or, to put it another way, to what extent does the global approach learn contexts that physically have no cause, and how does this affect generalization ability? After all, a global approach saves quite a bit of memory and avoids problems such as a shrinking domain due to convolution at each iteration step.

The approaches derived here are chosen to be as flexible as possible to apply to other problems. The data-based approach does not impose any limits on the modeling here, but rather offers the possibility to process measured data directly and to be trained with classical numerical solutions at the same time. Thus, even more complicated test cases can use this approach and give a starting estimate for the following time steps with relatively few time steps as training data and quite little training effort.

Achieving a converged, iterative procedure with coupled neural networks would be desirable in the future. However, this procedure does not exist yet and on the way there smaller and bigger stones have to be cleared out of the way. Until then, it is more promising to combine coupled neural networks and classical numerical solution methods in a hybrid approach, as suggested in other literature on this topic. Thus, the advantages of both methods can be used and the disadvantages can be compensated. The neural networks can be trained on the fly while the numerical solver computes the first time steps and later predict a good starting estimate for the solution in the next time step, which reduces the iterations of the numerical solver until convergence. Thus, computational effort can be saved and the neural network solution with lower accuracy can still lead to highly accurate results in the end.

If the training of the neural networks becomes more computationally intensive, there is also the possibility to realize a distributed training with PyTorch Distributed Data Parallel to get ready-to-use neural networks faster. This topic became relevant during the practical work but had to be put aside again as too far away from the relevant content after a few attempts. Nevertheless, there are still many possibilities to optimize the training with respect to the required time. Waiting times for results have to be bridged in a meaningful way somehow, less waiting time allows for a more focused investigation and delights everyone waiting for simulation results.

# 10 Conclusion

This work was intended to extend existing machine learning techniques to simulations of fluid-structure interaction. From the beginning, the literature on this topic showed that the potential was large on the one hand, but that existing solvers were not yet mature and needed support. To this end, physically informed neural networks and hybrid solvers came into closer view.
On the other hand, it had to be determined already here that the goal in this work is essentially to investigate possibilities of simulating fluid-structure interaction using neural networks and to describe challenges. A partitioned approach should be pursued, which trains a separate neural network for each physical subdomain and couples them together via the coupling library preCICE, in order to be able to observe the mutual interactions. Mathematically, this means solving a system of coupled partial differential equations. On the other hand, in a data-based approach, the equations are not of high relevance, the physics is to be learned through training examples.

In order to achieve this, single neural networks consisting of a convolutional layer, a long short-term memory and fully connected layers were implemented to account for both spatial and temporal connectivity correlations. These have shown that fluid flow and structural deformations, respectively, in one time step can be predicted using the same in the previous time steps. The results obtained showed a very good agreement with the reference solutions. However, fluctuations around the actual solution could not be completely avoided. With these results, new neural networks were developed that simulated a test case of fluid-structure interaction using the same principle. Here, each neural network was given the task of predicting the solution in its own domain based on the passed solutions in the whole domain. Each neuron's inputs were mapped to the outputs using a leaky ReLU activation function. A relative mean squared error loss was used along with the Adam optimizer to compute the network's weights. The preCICE coupling library was used to handle data communication and equation coupling in this partitioned approach. The results showed very good quality for both fluid flow and structural deformations in explicit coupling and exciting challenges in implicit coupling. In this work, the newly presented solution methods were only qualitatively investigated; the measurable time savings from this approach can be quantitatively investigated in future research on this topic.

Thus, this bachelor thesis, in which novel modeling approaches for fluid-structure interaction were considered from a simulation science perspective, ends with the realization that there is still much potential in the broad field of machine learning to extract information from data in such a way that it can be used profitably and with clear advantages over other solution methods. That the chosen approaches have their justification has become clear in the course of the work, that they cannot and should not replace classical solution methods either; rather, both can complement each other and the weaknesses of each solver can be transformed together into a powerful solver.

Summa summarum, we are in the beginning of a new era of artificial intelligence, in which there is still much to be discovered. I am convinced that the results of this work have unleashed a bit more of this outstanding potential.

# Bibliography

[Aba14] Abaqus. *Analysis User's Guide*. Simulia. Providence, 2014 (cit. on p. 38).

[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 Conference on Operating Systems Design and Implementation*. OSDI'16. Savannah, GA, USA: USENIX Association, 2016, pp. 265–283. ISBN: 9781931971331 (cit. on p. 30).

[BC09] O. A. Bauchau, J. I. Craig. "Euler-Bernoulli beam theory". In: *Structural Analysis. Solid Mechanics and Its Applications*. Vol. 163. Springer, Dordrecht, 2009. DOI: https://doi.org/10.1007/978-90-481-2516-6_5 (cit. on p. 38).

[BLG+16] H.-J. Bungartz, F. Lindner, B. Gatzhammer, M. Mehl, K. Scheufele, A. Shukaev, B. Uekermann. "preCICE – A fully parallel library for multi-physics surface coupling". In: *Computers and Fluids* 141 (2016). Advances in Fluid-Structure Interaction, pp. 250–258. ISSN: 0045-7930. DOI: https://doi.org/10.1016/j.compfluid.2016.04.003. URL: http://www.sciencedirect.com/science/article/pii/S0045793016300974 (cit. on pp. 13, 33, 49).

[BNK20] S. L. Brunton, B. R. Noack, P. Koumoutsakos. "Machine Learning for Fluid Mechanics". In: *Annual Review of Fluid Mechanics* 52.1 (2020), pp. 477–508. DOI: 10.1146/annurev-fluid-010719-060214. eprint: https://doi.org/10.1146/annurev-fluid-010719-060214. URL: https://doi.org/10.1146/annurev-fluid-010719-060214 (cit. on p. 15).

[Cho17] F. Chollet. *Deep Learning with Python*. Manning, Nov. 2017. ISBN: 9781617294433 (cit. on p. 16).

[Cyb89] G. Cybenko. "Approximation by superpositions of a sigmoidal function". In: *Mathematics of Control, Signals, and Systems (MCSS)* 2.4 (Dec. 1989), pp. 303–314. ISSN: 0932-4194. DOI: 10.1007/BF02551274. URL: http://dx.doi.org/10.1007/BF02551274 (cit. on p. 22).

[DBHV08] J. Degroote, P. Bruggeman, R. Haelterman, J. Vierendeels. "Stability of a coupling technique for partitioned solvers in FSI applications". eng. In: *COMPUTERS & STRUCTURES* 86.23-24 (2008), pp. 2224–2234. ISSN: 0045-7949. URL: http://dx.doi.org/10.1016/j.compstruc.2008.05.005 (cit. on pp. 51, 56).

[FMSC18] D. Ferras, P. A. Manso, A. J. Schleiss, D. I. C. Covas. "One-Dimensional Fluid–Structure Interaction Models in Pressurized Fluid-Filled Pipes: A Review". In: *Applied Sciences* 8.10 (2018). ISSN: 2076-3417. DOI: 10.3390/app8101844. URL: https://www.mdpi.com/2076-3417/8/10/1844 (cit. on p. 16).

[Gat14] B. Gatzhammer. “Efficient and Flexible Partitioned Simulation of Fluid-Structure Interactions”. Dissertation. München: Technische Universität München, 2014 (cit. on p. 51).

[GBB11] X. Glorot, A. Bordes, Y. Bengio. “Deep Sparse Rectifier Neural Networks”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by G. Gordon, D. Dunson, M. Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: http://proceedings.mlr.press/v15/glorot11a.html (cit. on p. 39).

[GBC16] I. J. Goodfellow, Y. Bengio, A. Courville. *Deep Learning*. http://www.deeplearningbook.org. Cambridge, MA, USA: MIT Press, 2016 (cit. on p. 16).

[HK20] R. Helmig, T. Köppl. *Dimensionsreduzierte Modellierung von Blutströmungen in großen Gefäßen. @articleJMLR:v15:srivastava14a, author = Nitish Srivastava and Geoffrey Hinton and Alex Krizhevsky and Ilya Sutskever and Ruslan Salakhutdinov, title = Dropout: A Simple Way to Prevent Neural Networks from Overfitting, journal = Journal of Machine Learning Research, year = 2014, volume = 15, number = 56, pages = 1929-1958, url = http://jmlr.org/papers/v15/srivastava14a.html*. Skript. Institut für Wasser- und Umweltsystemmodellierung, Universität Stuttgart, 2020 (cit. on p. 31).

[Hoc91] J. Hochreiter. “Untersuchungen zu dynamischen neuronalen Netzen”. Diplomarbeit. Institut fur Informatik der technischen Universität München, 1991 (cit. on p. 22).

[HSG+10] M. Hojjat, E. Stavropoulou, T. Gallinger, U. Israel, R. Wüchner, K.-U. Bletzinger. “Fluid-Structure Interaction in the Context of Shape Optimization and Computational Wind Engineering”. de. In: *Fluid Structure Interaction II*. Ed. by H.-J. Bungartz, M. Mehl, M. Schäfer. Vol. 73. Lecture Notes in Computational Science and Engineering. Springer Berlin Heidelberg, 2010, pp. 351–381. DOI: 10.1007/978-3-642-14206-2_13 (cit. on p. 49).

[KB17] D. P. Kingma, J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG] (cit. on pp. 24, 40).

[LBD+89] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, L. D. Jackel. “Backpropagation Applied to Handwritten Zip Code Recognition”. In: *Neural Computation* 1.4 (Winter 1989), pp. 541–551 (cit. on p. 21).

[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 (cit. on pp. 15, 29, 33, 41–43).

[SHK+14] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. In: *Journal of Machine Learning Research* 15.56 (2014), pp. 1929–1958. URL: http://jmlr.org/papers/v15/srivastava14a.html (cit. on p. 40).

[Tut21] preCICE Tutorials. *1 D Elastic Tube fluid-structure interaction case*. 2021. URL: https://precice.org/tutorials-elastic-tube-1d.html (visited on 01/26/2021) (cit. on pp. 51, 60).

**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