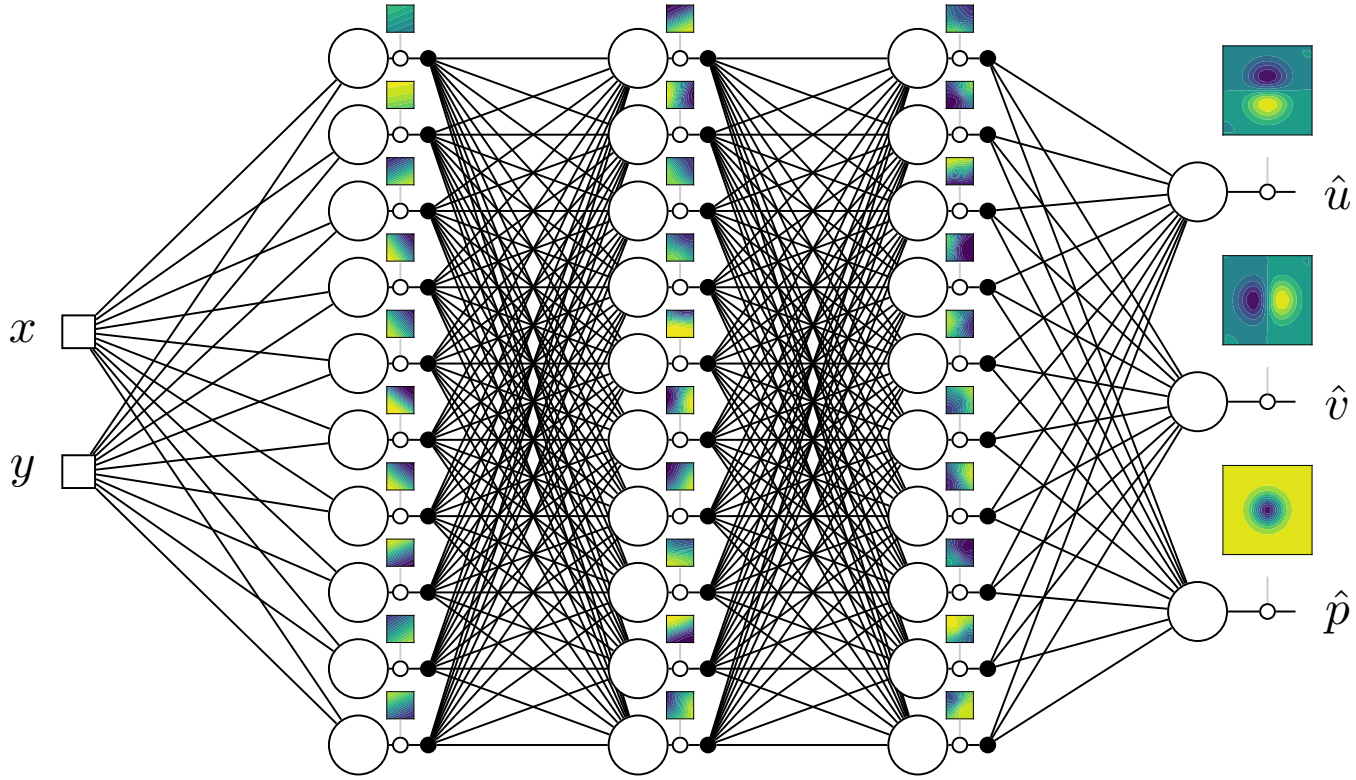
Master of Science Thesis

Artificial Neural Networks for Flow Field Inference

A machine learning approach

C.J. Terleth

2019



TU Delft

Artificial Neural Networks for Flow Field Inference

A machine learning approach

by

C.J. Terleth

to obtain the degree of Master of Science
at the Delft University of Technology,

Student number:	4377095	
Project duration:	September 27, 2018 – September 30, 2019	
Thesis committee:	Dr. R.P. Dwight,	TU Delft, supervisor
	Dr. ir. M. I. Gerritsma,	TU Delft
	Dr. ir. E. van Kampen,	TU Delft

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Abstract

Due to their powerful approximation capabilities, artificial neural networks have seen a wide interest in various fields. A particular application is the use of an artificial neural network to predict solutions of the governing equations for fluid flow i.e. the Navier-Stokes equations. This is done by taking the space and time variables as the inputs and the flow field variables (velocity and pressure) as outputs. Subsequently, the artificial neural network can be trained by using velocity field data on discrete points in a domain. This training is carried out by means of a scalar loss function, which is minimized by the backpropagation algorithm. By taking the gradients of the outputs with respect to the inputs, the Navier-Stokes can be formed and included in the same loss function. By including discrete velocity field data as well as the Navier-Stokes equations physics into the loss function, the trained artificial neural network could leverage its approximation capabilities to approximate the complex solutions of the flow variables. Only data on the velocity field is provided, such that the framework could be used to infer the pressure field, for which no data is given. In order to apply this approach to incompressible flows, the predicted velocity fields are strictly required to be divergence-free. To accomplish this, a divergence-free potential basis was proposed. This Data Physics Fluid Informed Neural Networks (DPFINN) framework was tested on several incompressible fluid flow test cases, for which analytical solutions were available. This work provides an extensive analysis for these test cases. The results of the test cases showed that the balance between the data and the physics loss function proved to be a delicate one, which depended on a number of different choices for the artificial neural network architectures, data and physics informing process as well as the training process. Overall, the flow fields were able to be inferred well, however simultaneous implementation of the data and the physics proved to be difficult. More complex artificial neural networks were shown to not always improve the performance. Thus, the framework results were showed to be largely dependent on the balance between the expressibility and the trainability of the framework.

Acknowledgments

After my Bachelor studies in Amsterdam, I set out a new personal challenge, to obtain a Master's degree within the field of Aerospace Engineering. Specifically, in the field that interested me the most: Aerodynamics. In order to be admitted to the program, I had to complete a bridging program first. Which proved to be a proper first test in the sense of many new concepts which were presented to me in difficult order. The Master's program improved in that sense, however not in the sense of the amount of work required to even pass the smallest credited courses. It proved to be a challenge that I was looking for and perhaps more.

To the reader, it may seem that I have completed this challenge purely by myself but this could not be further from the truth. Many people deserve my gratitude as many people have supported me throughout my journey here in Delft. Surely too many for me to remember or mention, nevertheless a thank you to all of them is in order.

However, a personal big thank you goes out to my parents. Their unconditional support as well as creating a family based on love but also practicality has helped me thrive throughout my life so far. To them, I would also like to say: Thank you, your goal of helping both me and my sister to a good education is hereby fulfilled and you could not have done a better job at it. Gratitude also goes to my sister, Maaike, who always actively supported me and was always there for me when I needed her.

One of the life lessons that my parents have given me is to balance out work/studies with hobbies/sports well. Therefore, almost immediately after arrival in Delft, I picked up my old sport, baseball by joining HitManics, which is a baseball- and softball club for and run by students of the university. It has enjoyed a big growth and has changed a lot during my time there. That is all because the club contains a great collection of very nice people, to whom I would all like to say thank you. Without you and baseball, my balance would be off. A special mention goes to my team as well as the new friends I made there, with special mentions for Thijs, Anouk, Tung, and Mauryze.

Jaime referred to it well as he said that the basement is a dark place but a fantastic workplace for carrying out a Master thesis. It is a rather large space as well, so too many people to thank that contributed to the nice international atmosphere in there (pun intended). Nevertheless a special thank you goes out to my closest neighbors: Jaime, Carlos, Javier as well as to Jorge and Dorian.

Last but not least, I would like to thank my supervisor Dr. Richard Dwight for giving me the opportunity to carry out my thesis, providing me valuable resources and giving me the freedom and support throughout my thesis.

Niels Terleth
Delft, September 2019

Contents

List of Figures

1

Introduction

In the field of Aerodynamics, a distinction is made between experimental and computational aerodynamics. Experimental aerodynamics relates to studying flow fields in controlled physical environments, generally with the goal of studying the flow field around a particular object. Computational aerodynamics concerns itself with modeling fluid flows within a computational domain. Herein, the governing equations - the Navier-Stokes equations - are discretized and approximated on a computational grid, which forms the definition of Computational Fluid Dynamics (CFD). In industry, the experimental side and the computational side of aerodynamics generally complement each other in design tasks of aerodynamic objects while the experimental side also serves a purpose in validating CFD codes. There has been academic work carried out in where the governing equations are used in the experimental field on a grid in order to obtain flow variables of interest e.g. pressure fields [40]. However, utilizing experimental data for informing CFD solutions, is a less trivial task.

Due to recent developments in algorithms and hardware (in particular graphical processing unit (GPU) developments), the machine learning (ML) field has seen increased popularity. Machine learning is the field where computer models are trained in order to increase the performance on a particular designated task. The applications vary and are widespread, for example machine learning has been used for image classification or regression problems successfully. One model of particular interest is the artificial neural network, which is vaguely inspired by the biological neural networks that constitute animal/human brains. By training the artificial neural network, the neural network can in principle approximate any (unknown hidden) functional provided the artificial neural network is complex enough [19]. Therefore, modeling physical phenomena by means of neural networks is an interesting application.

The aforementioned computational field of aerodynamics aims to model flow physics in the form of CFD. However, a recent industry survey [16] with regard to the future of CFD showed that e.g. accurately predicting turbulent (separated) flows remains an issue. It is therefore of importance to keep exploring alternatives or novel ideas within the fluid flow academic field. For example, by combining the information that is readily available i.e. combining data in the form of flow experiments and physics in the forms of the Navier-Stokes equations. A possible means to do this with is through machine learning. More specifically, by using artificial neural networks that takes as inputs the coordinates and as outputs the flow variables can be employed. By differentiating the outputs with respect to the inputs, the Navier-Stokes equations can be formed. Using these, in conjunction with the discrete velocity field data, as a target for the artificial neural network to be learned to, the flow variables can be found in a data-assimilation fashion. A particular application of such a framework, can be to the problem of finding pressure from Particle Image Velocimetry (PIV) velocity fields (see e.g. [47] and [46]). In order to asses the potential of such an approach, the framework first needs to be tested to simple (analytical) flow cases. This is where this thesis aims to contribute to.

1.1 Background

The problem of pressure from velocity fields is one that dates back as far as 1935 by the work of [41], however most of the approaches use either a method that integrates the pressure gradients obtained from evaluation of the Navier-Stokes equations or by solving a Pressure-Poisson equation ([48] and [13]). The approach of using artificial neural networks uses neither. It combines the data and physics in a least-squares approach, where the idea is that by combining both for learning a complex artificial neural network, velocity as well as pressure can be inferred from velocity data. For incompressible flows, the mass conservation equation reduces to the divergence of velocity being equal to zero. Therefore, mathematically, for incompressible flows, the divergence of the velocity is required to be zero. Creating divergence-free solutions in the data-assimilation context has seen different approaches. For example, one by [4], where a basis is utilized to enforce the resulting velocity field being divergence free. The same basis can be used in the context of the artificial neural networks data assimilation approach for fluid flow problems.

The foundation for the artificial neural network data-assimilation approach lays in the works of [36], [37] and [37]. Here, Physics-Informed Neural networks are employed to solve partial-differential equations such as the Burger's equation and the Navier-Stokes equations. More specifically, [37] considers wake regions behind cylinders. [8] and [7] apply the same approach to Stokes problems and electroosmotic flows in microchannels. However, none of these works give a thorough analysis of implications on the results by different choices of artificial neural networks or parameters. About two decades before, similar efforts were done for solving Ordinary Differential Equations e.g. as in [28] and [29]. A large difference between the earlier and the more recent works is the implementation of the boundary conditions. In recent works, a least-squares approach is used, whereas the earlier works used trial functions to enforce the boundary conditions. Other recent works for solving general partial differential equations (PDE) can be found, see for example [43], [27], [10].

Almost all of these works use least-squares minimization of a scalar loss function for the training of the artificial neural network and this least-squares approach is key to this artificial neural network approach and can be comparable to the approaches used in least-squares Finite Element Method (LS-FEM), see for example [22], [23] or [42]. For the least-squares approach, the domain of interest is sampled by discrete points. On these discrete points, the governing equations are to be minimized. The same applies to the data, they are available on discrete points in the domain, and therefore a least-squares minimization target (loss function) can be employed for the discrepancy between the artificial neural network output and the data. Both the data and physics loss functions are combined to form one loss function and the choices available in forming the loss functions are great, while the implications of each choice often unclear. Nevertheless, because of this least-squares approach, the domains that can be considered do not need to have an easy shape. No mesh or grid is to be constructed. Therefore, any shaped domain can be considered. This was already mentioned and done by [29] and later on again by [9]. Moreover, because of the least-squares approach the implementation of (Dirichlet) boundary conditions and data into the approach is in principle the same. To conclude, the velocity fields generated by such least-squares approaches often lack mass conservation [42], to which usage of a divergence-free basis could provide a remedy.

1.2 Research Objectives

The main objective of this research is to formulate the data physics fluids informed neural networks framework. The fluid problems for which the framework will be tested on, are incompressible. Therefore, part of this objective is to include a method into the framework such that the inferred velocity fields are divergence-free, similarly as is done by [4]. Next to inference of the velocity fields, the pressure fields are to be inferred as well, by means of the velocity field data and the physics. This will provide an alternative method for the existing methods currently available for pressure from velocity field problems . The physics implementation needs to be done consistently and needs to be converted to a least-squares minimization target in the form of a loss function that is to be used in the artificial neural network training process.

In order to test the flow field inference capabilities, the framework is tested extensively for a vari-

ety of test cases, networks and other parameters. For the implementation of the physics into the loss function, that augments the data provided to the loss function, several forms are available. Commonly, in fluid mechanics, the governing equations are non-dimensionalized. Therefore, already two forms of governing equations are available, the dimensional and the non-dimensional one. For artificial neural networks, according to [35] and [38] it is recommended to have inputs that are in a range that is $[-1, 1]$ (normalization), this is seldom the case for fluid flow problems i.e. domains can stretch several times the reference length scale in each spatial direction. Therefore, by applying scaling, a new form of physics can be obtained yielding three possible forms for the physical equations. The reason for the normalization is because it enhances the learning algorithm's performance. Thus, the second objective is to investigate which of those three physics forms produces the best results for learning the artificial neural network and the inference of the flow variables.

1.3 Research Questions

The main objectives are formulating the data physics informed neural networks framework with the divergence-free velocity field inference capability as well as assessing the velocity and pressure field inference performance of the framework. For the assessment of the performance of the framework, several different artificial neural networks and learning strategies can be used which give rise to the first research question:

Main Research Question 1: *How can artificial neural networks be used to infer flow fields from given velocity field data and flow physics?*

- **Research question 1:** *Which artificial neural networks are suitable for fluid flow data assimilation purposes?*
  The artificial neural network model will be used for this project. However, the network architecture can vary greatly and many different types of architectures are available.

- **Research question 2:** *Which loss function choices are the best for the implementation of the data and physics?*
  Because multidimensional problems are considered and the training of the artificial neural network requires a scalar function, the data as well and the physics are to be reduced to that particular scalar loss function. The reduction of both can be done in various ways by means of different loss function types.

- **Research question 3:** *Which machine learning algorithms are appropriate for learning artificial neural networks for flow field inference?*
  After a network has been chosen and a loss function is formulated, a suitable learning algorithm has to be chosen that is efficient in computational resources as well as robust in terms of results consistency. It is unknown how dependent the choice of a learning algorithm is on the loss function formulation.

More specifically, the implementation of the physics gives rise to the second main research question:

Main Research Question 2: *How can the Navier-Stokes equations including mass conservation be implemented in the framework?*
Or in other words, how can a physically consistent divergence-free velocity field be obtained from the framework?

- **Sub-question 1:** *What is the best formulation of the Navier-Stokes equations to be used in the loss function?*
  The Navier-Stokes equations can either be formulated in non-dimensional or dimensional form or scaled differently, which is possibly more suitable for artificial neural networks.

- **Sub-question 2:** *How can the requirement of the divergence-free velocity fields be satisfied by the framework?*

1.4 Thesis Outline

This work aims to continue the work on the data-physics informed neural networks with special application to flow problems. Therefore, first the artificial neural networks for the framework are explained. With special attention to the way the artificial neural network forms the outputs as well as how the gradients are computed (Chapter 2). Secondly, the data-physics informed neural networks framework is discussed (Chapter 3). Here, a general formulation is given especially with regard to the implementation of the governing equations as well as the consistent scaling of them. More specifically, the choices and the way the framework is coded is discussed here. Testing of the framework is carried out on a selection of test cases, which are discussed in Chapter 4. Here also, the exact formulations of the Navier-Stokes equations forms that are used are given. Moreover, the divergence-free implementation is also discussed here, in order to preserve framework generality in Chapter 3. Lastly, the fluid flow test cases are utilized to test the performance of the data-physics fluids informed neural networks in Chapter 5, after which the research conclusions are summarized in Chapter 6.

2

Artificial Neural Networks

Within Machine-Learning (ML), a commonly employed model is the artificial neural network (ANN). Just like a biological brain, the idea behind the ANN is to arrange multiple artificial neurons (Section 2.1) in layers to form an artificial neural network (Section 2.2). This will form a complex mapping between inputs and outputs. If the mapping that forms the artificial neural network is complex enough for a given approximation problem, it could be referred to as a universal approximator (Section 2.3). However, this is largely dependent on the learning capabilities of the artificial neural network. A more complex artificial neural network does not necessarily give better results. Learning the artificial neural network is done by means of advanced gradient-descent methods and therefore require gradient information (Sections 2.4 and 2.5).

2.1 Artificial Neurons

The artificial neurons found in artificial neural networks are loosely inspired by the biological ones found in e.g. humans. The neurons found in human brains consist out of three components: the dendritic tree, the cell body, and the axon [38]. The dendritic tree forms the connection with other neurons and collects signals from them. The cell body integrates the signals and generates an output signal. Subsequently, the output is passed on to other neurons through the branching axon. The artificial neuron or perceptron works similarly according to [38]. Inputs are fed through the computing node through connections with other neurons, which are then processed through summation and an activation function to form an output. This output can then be distributed to other neurons (Figure 2.1). This artificial neuron model is occasionally referred to as the McCulloch [31] model or perceptron.

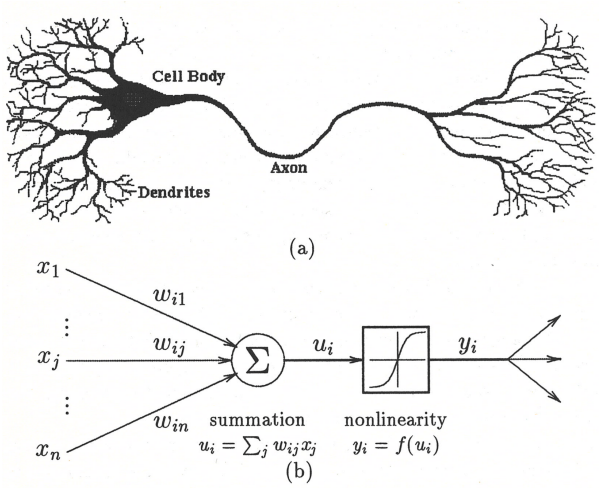


Figure 2.1: (a) Biological neuron. (b) artificial neuron [38]

The artificial neuron can be seen as an extremely crude model of a biological neuron. The biological neuron, in reality, is much more complex, so the resemblance is little. The artificial neurons commonly employ simplified activation functions, that process the inputs. An overview of common activation functions can be found in Figure 2.2. Some functions are non-linear or discontinuous. Generally, the magnitude of the output is $|\sigma(s)| \leq 1$. Activation functions are sometimes also referred to as squashing functions or limiters since they have the ability to limit/convert large input values to smaller output values. Activation functions such as the sigmoid or step function also limit the output to only positive values. The most common activation functions are the sigmoid logistic function and the hyperbolic tangent (tanh), due to their smooth and non-decreasing properties. Because of their inherent smoothness, these activation functions are also occasionally referred to as being soft limiters, where discontinuous activation functions are referred to as hard limiters [38].
Based on Figure 2.1(b), the artificial neuron signal processing can be divided into four parts, as is also done by [35].

1. **Weighting:** Incoming signals x from other neurons are each weighted by a constant scalar (typically denoted by w).

2. **Summation:** Weighted linear combination of incoming signals is formed.

3. **Biasing:** An additional constant factor is added to the linear combination, which is referred to as the bias (b).

4. **Activation:** the scalar quantity resulting is then fed through the activation function which then forms the output.

This can be expressed mathematically by summation of vector products. For example, for a single artificial neuron with N inputs, the equation that forms the output could be expressed in the following way:

Hyperbolic tangent function

$$\sigma(s) = \tanh(s)$$
$$\sigma'(s) = \sigma(s)\,(1 - \sigma(s))$$

Sigmoid function

$$\sigma(s) = 1/(1 + e^{-s})$$
$$\sigma'(s) = 1 - \sigma(s)^2$$

Step function

$$\sigma(s) = \begin{cases} 1 & s \geq 0 \\ 0 & s < 0 \end{cases}$$
$$\sigma'(s) = \delta(s)$$

Sign function

$$\sigma(s) = \begin{cases} 1 & s \geq 0 \\ -1 & s < 0 \end{cases}$$
$$\sigma'(s) = 2\delta(s)$$

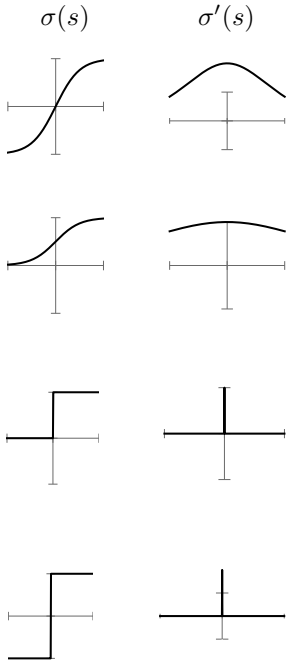
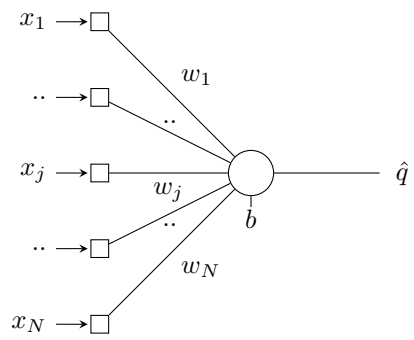
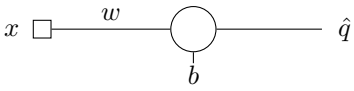


Figure 2.2: Common artificial neuron activation functions used for artificial neural networks, adapted from [38]

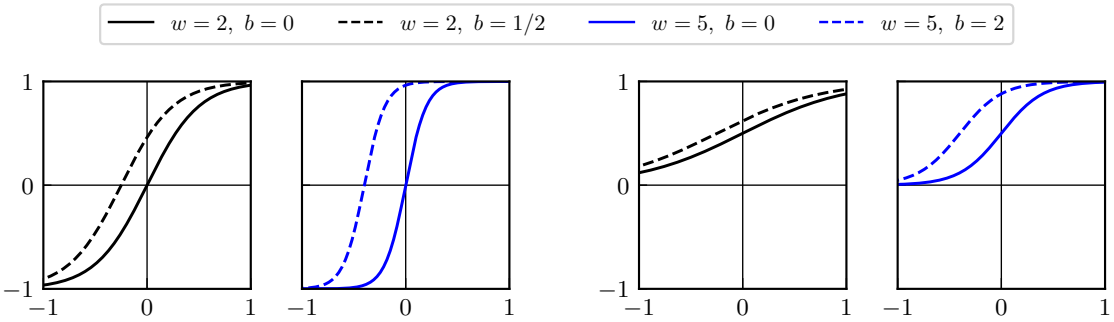


$$z = \sum_{j=1}^{N} w_j x_j + b \qquad (2.1)$$
$$\hat{q} = \sigma(z)$$

Where the inputs are denoted by x_j for $i = 1, ...N$ and the corresponding weights are denoted as w_j. The bias is denoted by b. Due to graphical complexity for larger artificial neural networks. The summation, as well as the activation, is drawn within the artificial neuron (drawn as a circle). The square nodes for the inputs do not contain any processing or can be seen as artificial neurons with linear activation function, unit weights and without bias. A single artificial neuron divides the input space ($\mathbb{R}^N$) into two parts by a hyperplane. The hyperplane, which is a line in two-dimensional space and a plane in higher-order space, where the order of the space is governed by the number of inputs N. The combination of weights and bias for an artificial neuron form the basis of the hyperplane. The weights govern the orientation of the hyperplane whereas the bias shifts the hyperplane in the direction of the weight vector $\mathbf{w}$. The amount of shift is $d = -\frac{b}{||\mathbf{w}||}$ [38]. For just a single input (x) the single weight (w) acts as a slope parameter [21], (Figure 2.3) i.e. when w is large for a *tanh* activation function, the activation function approximates the *sign* activation function from Figure 2.2. The same is true for higher dimensions, perpendicular to the orientation of the hyperplane. For the step and the sign activation function, the weight does not affect the slope (at the hyperplane), since it is fixed and infinite, but the bias still shifts the hyperplane in the direction of the weight vector.
When a single neuron has two inputs e.g. x_1 and x_2, there are two weights w_1 and w_2. The vector $\mathbf{w} = [w_1, w_2]^T$ defines the orientation of a line in 2D space, the magnitude of the weight vector defines the slope of the activation function. The bias and the magnitude of the weight vector govern the shifting of the hyperplane. The distance perpendicular to the line is then the input to the artificial neuron. As



(a) Single artificial neuron with one input w and bias b

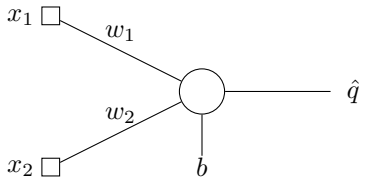


(b) Hyperbolic tangent activation function $\sigma(s) = \tanh(s)$

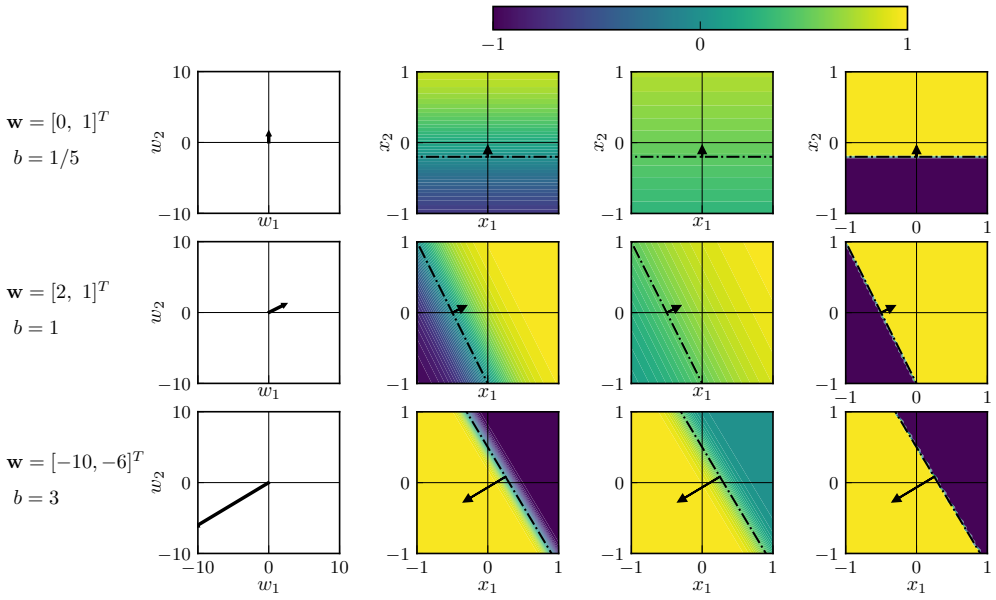
(c) Sigmoid activation function $\sigma(s) = 1/(1 + e^{-s})$

Figure 2.3: output of single artificial neuron with one input (x) and tanh or sigmoid activation function. Various weights bias combinations, $\hat{q} = \sigma(wx + b)$

also described by [38], by dividing the input space, the artificial neuron acts as a classifier, e.g. data vectors for which the inner product with the weight vector is positive will result in positive outputs. Figure 2.4 shows the output of an artificial neuron with two inputs for various combinations of weights and bias constants and different activation functions. The magnitude of the weight vector is largest for the bottom column, which also results in a stronger threshold boundary than for example the weight vector in the top one. This is only the case for the tanh and sigmoid activation functions. The effect of bias is also clear by the fact that the thresholds are shifted by a factor of $-b/||\mathbf{w}||$.



(a) Single artificial neuron with two inputs, $\mathbf{w} = [w_1, w_2]$ and bias b

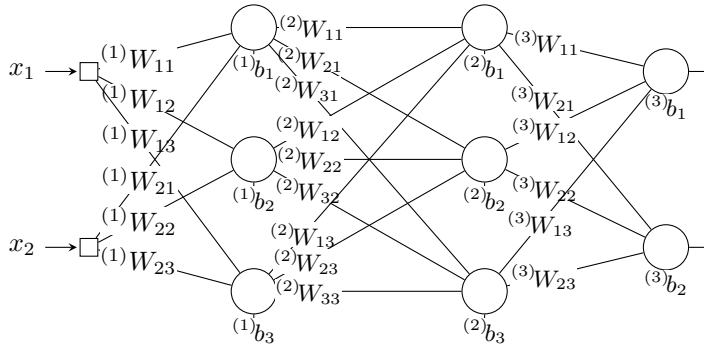


(b) weights and biases (c) $\sigma = \tanh$ (d) $\sigma = 1/(1 + e^{-s})$ (e) step function

Figure 2.4: Single neuron output for $N = 2$ inputs, for various activation functions as well as different weights and bias combinations

2.2 Networks

By adding neurons in layers, artificial neural networks are formed. More layers and more artificial neurons increase the complexity of the artificial neural network, as for each layer, all the artificial neurons are connected to all the ones in the next layer, where each connection contains a weight and each artificial neuron has a bias and a predefined activation function. The combination of all the weights, biases in the network and the activation function(s) of the artificial neurons determine the size and shape of the output. Such a network is shown below. Although uncommon practice, the weights, and biases are shown for the connection and artificial neurons respectively. The network has one input layer, two hidden layers, and one output layer. The input layer is just for the inputs and does not contain any activation. The hidden layers contain weights and biases and the artificial neurons have activation functions ($\sigma(s)$). The hidden layers are referred to as hidden not because they can not be accessed but because the effect of the individual artificial neurons on the outputs is often hard to estimate. The output nodes have linear activation functions but also contain weights and biases. The mathematical expressions are shown in equation 2.2 and are in vector-form, where the W are tensors, the inputs are vectors $\mathbf{x}$ as well as the outputs $\hat{\mathbf{q}}$ and the biases $\mathbf{b}$. The superscript in front of the symbols denote the layer $l = 1, \ldots, La + 1$. The weight tensor entries represent the weight between connecting nodes. The subscript indices of ${}^{(l)}W_{ij}$, i and j denote the node it is pointing to and coming from respectively. So, for the connection between the input node for x_2 and the first node in the first hidden layer, the weight is ${}^{(1)}W_{12}$. For the biases, the subscript just denotes the bias for their respective nodes. Furthermore, for each layer, the summation and bias step yield $\mathbf{z}$. The application of the activation function σ to $\mathbf{z}$ (element-wise), will yield $\mathbf{a}$, which is the output of the artificial neurons in that layer and is what will be used for weighting and addition of the bias for the next layer.



$$
\begin{aligned}
{}^{(1)}\mathbf{z} &= \mathbf{x}\left({}^{(1)}W\right)^T + {}^{(1)}\mathbf{b} \\
(1)\mathbf{a} &= \sigma\left({}^{(1)}\mathbf{z}\right) \\
{}^{(2)}\mathbf{z} &= {}^{(1)}\mathbf{a}\left({}^{(2)}W\right)^T + {}^{(2)}\mathbf{b} \\
{}^{(2)}\mathbf{a} &= \sigma\left({}^{(2)}\mathbf{z}\right) \\
\hat{\mathbf{q}} &= {}^{(2)}\mathbf{a}\left({}^{(3)}W\right)^T + {}^{(3)}\mathbf{b}
\end{aligned}
\tag{2.2}
$$

$$
\mathbf{x} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \qquad \hat{\mathbf{q}} = \begin{pmatrix} \hat{q}_1 & \hat{q}_2 \end{pmatrix}
$$

$$
{}^{(1)}W = \begin{pmatrix} {}^{(1)}W_{11} & {}^{(1)}W_{12} \\ {}^{(1)}W_{21} & {}^{(1)}W_{22} \\ {}^{(1)}W_{31} & {}^{(1)}W_{32} \end{pmatrix} \qquad {}^{(1)}\mathbf{z} = \begin{pmatrix} {}^{(1)}z_1 & {}^{(1)}z_2 & {}^{(1)}z_3 \end{pmatrix}
$$

$$
{}^{(1)}\mathbf{b} = \begin{pmatrix} {}^{(1)}b_1 & {}^{(1)}b_2 & {}^{(1)}b_3 \end{pmatrix} \qquad {}^{(1)}\mathbf{a} = \begin{pmatrix} {}^{(1)}a_1 & {}^{(1)}a_2 & {}^{(1)}a_3 \end{pmatrix}
$$

$$
{}^{(2)}W = \begin{pmatrix} {}^{(2)}W_{11} & {}^{(2)}W_{12} & {}^{(2)}W_{13} \\ {}^{(2)}W_{21} & {}^{(2)}W_{22} & {}^{(2)}W_{23} \\ {}^{(2)}W_{31} & {}^{(2)}W_{32} & {}^{(2)}W_{33} \end{pmatrix} \qquad {}^{(2)}\mathbf{z} = \begin{pmatrix} {}^{(2)}z_1 & {}^{(2)}z_2 & {}^{(2)}z_3 \end{pmatrix}
$$

$$
{}^{(2)}\mathbf{b} = \begin{pmatrix} {}^{(2)}b_1 & {}^{(2)}b_2 & {}^{(2)}b_3 \end{pmatrix} \qquad {}^{(2)}\mathbf{a} = \begin{pmatrix} {}^{(2)}a_1 & {}^{(2)}a_2 & {}^{(2)}a_3 \end{pmatrix}
$$

$$
{}^{(3)}W = \begin{pmatrix} {}^{(3)}W_{11} & {}^{(3)}W_{12} & {}^{(3)}W_{13} \\ {}^{(3)}W_{21} & {}^{(3)}W_{22} & {}^{(3)}W_{23} \end{pmatrix} \qquad {}^{(3)}\mathbf{z} = \begin{pmatrix} {}^{(3)}z_1 & {}^{(3)}z_2 & {}^{(3)}z_3 \end{pmatrix}
$$

$$
{}^{(3)}\mathbf{b} = \begin{pmatrix} {}^{(3)}b_1 & {}^{(3)}b_2 & {}^{(3)}b_3 \end{pmatrix} \qquad {}^{(3)}\mathbf{a} = \begin{pmatrix} {}^{(3)}a_1 & {}^{(3)}a_2 & {}^{(3)}a_3 \end{pmatrix}
$$

The four parameters that govern the size and shape of the artificial neural networks are: N_i (the number of inputs), N_n (the number of artificial neurons in each hidden layer, excluding output layer), N_o (the number of outputs) and finally La, the number of hidden layers (Figure 2.5). Since the output layer can be seen as a hidden layer as well, often the output layer is included in the definition in the number of

hidden layers. However, throughout the remainder, La is strictly the number of hidden layers excluding the output layer. Next to these four parameters, the activation function of the artificial neurons is an additional degree of freedom. These five parameters characterize the artificial neural network.

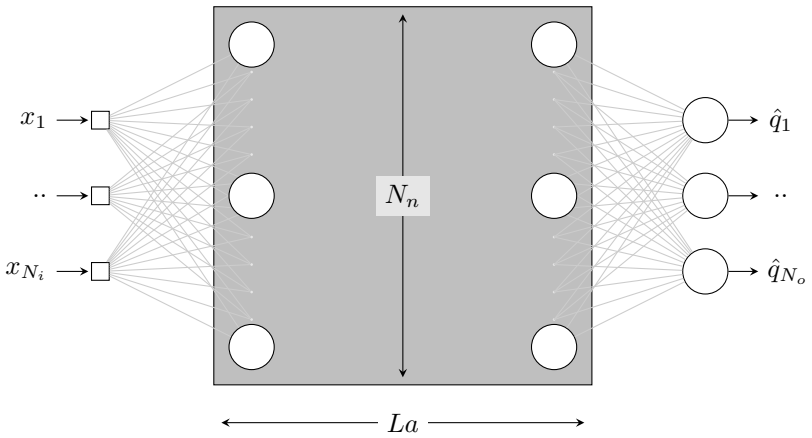


Figure 2.5: General artificial neural network parameters (N_i, N_n, N_o, La) schematic. The hidden layers are drawn in the gray box.

It is worthwhile to devise a general formulation for the output of an arbitrary artificial neural network. The inputs $\mathbf{x}$ is in $\mathbb{R}^{1\times N_i}$ and is a column vector. The first weight tensor will then be ${}^{(1)}W \in \mathbb{R}^{N_n\times N_i}$. The multiplication of the input column vector $\mathbf{x}$ and the first weight tensor as well as summation with the bias vector, ${}^{(1)}b \in \mathbb{R}^{1\times N_n}$ will yield ${}^{(1)}\mathbf{z} \in \mathbb{R}^{1\times N_n}$, also a column vector. The output of the first layer will be another column vector ${}^{(1)}\mathbf{a} \in \mathbb{R}^{1\times N_n}$. The propagation of ${}^{(1)}\mathbf{a}$ through the second hidden layer involves a tensor product with the next weight tensor ${}^{(2)}W \in \mathbb{R}^{N_n\times N_n}$ and summation with ${}^{(2)}\mathbf{b} \in \mathbb{R}^{1\times N_n}$, which will then be fed through the activation function. This continues until all the La number of hidden layers have been passed resulting in ${}^{(La)}\mathbf{a} \in \mathbb{R}^{1\times N_n}$. This will be weighted again (by ${}^{(La+1)}W \in \mathbb{R}^{N_o\times N_n}$) and biased (by ${}^{(La+1)}\mathbf{b} \in \mathbb{R}^{1\times N_o}$) to form the output $\hat{\mathbf{q}} \in \mathbb{R}^{1\times N_o}$. This can be expressed more generally and briefly by equation 2.3.

$$
\begin{aligned}
{}^{(0)}\mathbf{a} &= \mathbf{x} \\
{}^{(1)}\mathbf{z} &= {}^{(0)}\mathbf{a}\left({}^{(1)}W\right)^T + {}^{(1)}\mathbf{b} \\
{}^{(1)}\mathbf{a} &= \sigma\left({}^{(1)}\mathbf{z}\right) \\
&\cdots \\
{}^{(i)}\mathbf{z} &= {}^{(i-1)}\mathbf{a}\left({}^{(i)}W\right)^T + {}^{(i)}\mathbf{b} \\
{}^{(i)}\mathbf{a} &= \sigma\left({}^{(i)}\mathbf{z}\right) \\
&\cdots \\
\hat{\mathbf{q}} &= {}^{(La)}\mathbf{a}\left({}^{(La+1)}W\right)^T + {}^{(La+1)}\mathbf{b}
\end{aligned}
\tag{2.3}
$$

As vectors can mathematically, be considered tensors the input tensor $\mathbf{x}$ is fed through the artificial neural network and during this will undergo transformations and scalings according to the weights and biases. For simple networks, these operations are understandable, however, this rapidly changes when the number of artificial neurons and/or hidden layers increase. As an example, a simple artificial neural network ($N_n = 2$, $N_i = 2$, $La = 1$, $N_o = 1$) is considered. A 2D domain $\mathbf{x} = (x, y)$ is fed through the network. The weights connecting the inputs to the first hidden neuron are: ${}^{(1)}W = \begin{pmatrix}4 & 0\end{pmatrix}$. Therefore, the x-inputs are favored and scaled by 4, where the y-inputs are scaled to zero. When fed through the first neuron - which has a tanh activation function - the result is a 2D function that is solely and strongly dependent on x, as the bias is set zero. For the second hidden neuron, the weights are equal

for both inputs: ${}^{(2)}W = \begin{pmatrix} 1 & 1 \end{pmatrix}$ and the bias is zero. This results in a hyperplane that runs diagonally through the origin of the domain, where the largest output is in the top right corner and the lowest in the bottom left corner. With equal weighting of both neuron outputs (${}^{(3)}W = (1, 1)$), and a zero bias, the output $\hat{q}$ of the network is a smooth function with a smoothened hyperplane around the diagonal.

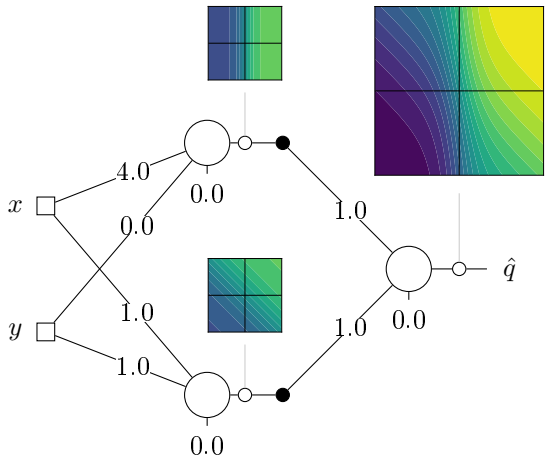


Figure 2.6: Simple artificial neural network example: ($N_n = 2$, $N_i = 2$, $La = 1$, $N_o = 1$, $\sigma = tanh$). ${}^{(1)}W = (4, 0)$, ${}^{(2)}W = (1, 1)$, all biases zero.

For more complex networks, going through the artificial neural network, by looking at each neuron output becomes rather cumbersome and is somewhat uninformative. An example of such an attempt is shown in Figure 2.7. Here weights are randomly chosen as well as biases. By examining closely, some features in the output can be traced back to the outputs of individual neurons. For example, the outputs seem to decrease or increase in the same specific directions. This is, however, considered a small network, for larger networks, generating similar figures and examining becomes significantly harder. Also from equation 2.3 it can be said that the weights of the connections between the last hidden layer and the output layer provide scaling while the biases in the output layer merely add or subtract to the output.

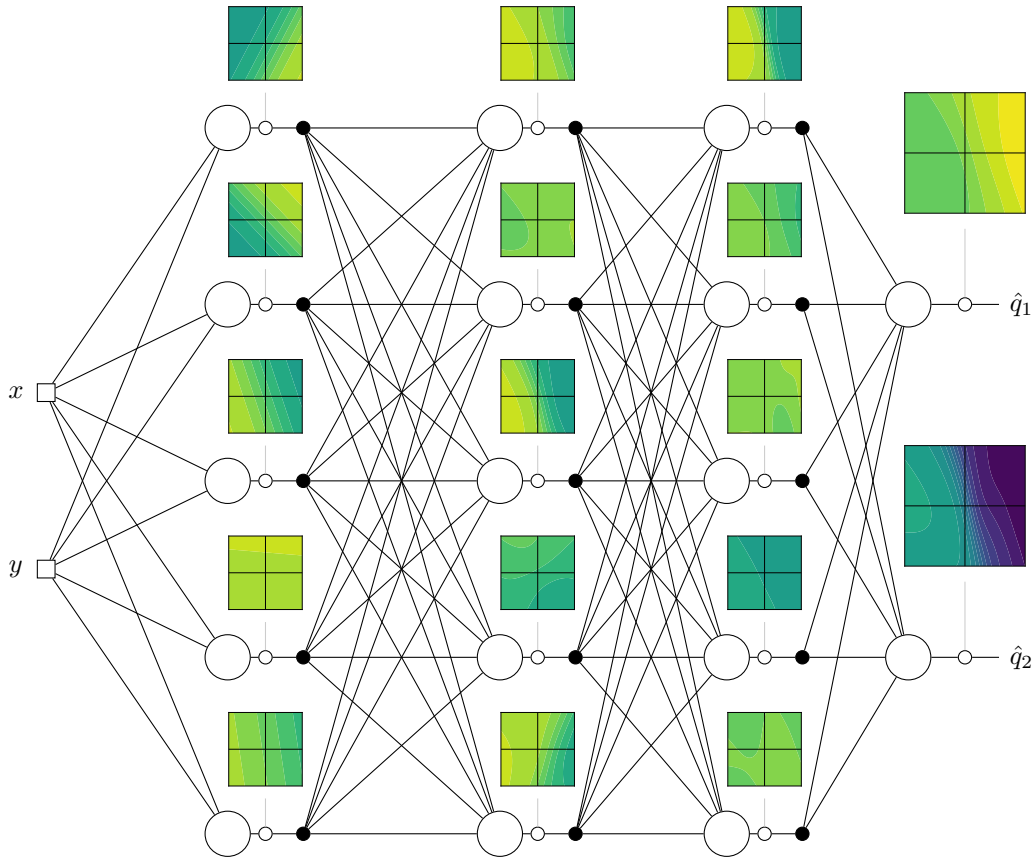


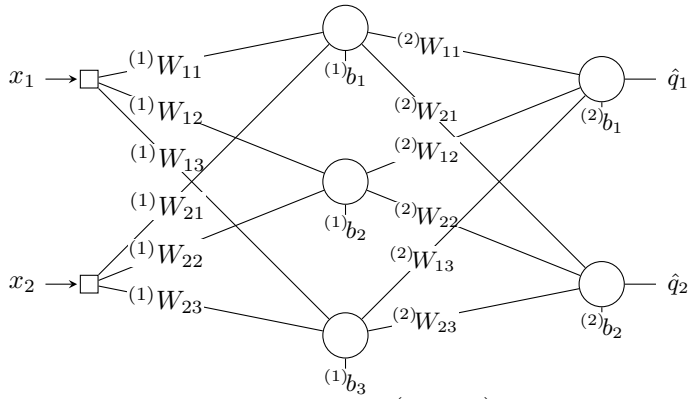
Figure 2.7: Complex artificial neural network example: ($N_n = 5$, $N_i = 2$, $La = 3$, $N_o = 2$, $\sigma = tanh$).

2.3 Approximation Capabilities

Artificial neural networks are seen as powerful tools for tasks such as pattern classification or function approximation, but throughout the development of algorithms and models, the question of how well models can represent data sets is of paramount importance. Literature such as [38], [17] or scientific work such as [37] that utilize artificial neural networks refer to works such as [19] or [26] in order to justify using artificial neural networks. These works, in turn, refer to [25] which is occasionally referred to as the Kolmogorov superposition theorem or universal approximation theorem. This property can be stated as done by [19]: *"There is a single hidden layer feed-forward neural network that approximates any measurable function to any desired degree of accuracy."* [26] takes a different route by suggesting: *"If the polynomial P approximates the function f with precision $\epsilon/2$ and a neural network approximates P with the same precision, then this network approximates f with the desired precision ϵ."*. [20] suggests that single hidden layer neural networks are not always effective if the number of artificial neurons in the hidden layer is prescribed, but that this is not the case for artificial neural networks with multiple hidden layers. However, none of these theorems give any practical insight into the choice of activation functions or e.g. the minimal number of neurons required to approximate the hidden function with the desired accuracy. It just proves the existence of solutions for artificial neural networks with single hidden layers. Finding the right artificial neural network architecture is still an iterative task, which is dependent on how complex the underlying latent function is as well as the ability to train the artificial neural network.

2.4 Differentiation

Of importance are the gradients of the artificial neural networks. For training but also the gradients of the outputs with respect to the inputs. By using the chain-rule, these gradients can be computed. For example, let there be an artificial neural network ($N_n = 3$, $N_i = 2$, $La = 1$, $N_o = 2$). The gradients of that artificial neural network can be computed by applying the differentiation chain-rule, as the output can be expressed by a single equation as in equation (2.5). In essence, the variables of the outputs are the inputs and all the weights and biases that are used. Differentiation with respect to a certain variable is then simply going from outside to inwards in the full equation (2.5) until the variable is reached. As an example, the outputs are expanded. From the equations it is clear that to generate the output of the first hidden layer, all the weights of ${}^{(1)}W$ and all the biases of ${}^{(1)}\mathbf{b}$ are used. To go from the outputs of that hidden layer to the first output of the network output $\hat{q}_1$, the outputs and weights from all the hidden artificial neurons in the hidden layer to the first output $\hat{q}_1$ are used i.e. the first row of ${}^{(2)}W$. This output is biased by the ${}^{(2)}b_i$



$$
\begin{aligned}
{}^{(1)}\mathbf{z} &= \mathbf{x}\left({}^{(1)}W\right)^T + {}^{(1)}\mathbf{b} \\
(1)\mathbf{a} &= \sigma\left({}^{(1)}\mathbf{z}\right) \\
\hat{\mathbf{q}} &= {}^{(1)}\mathbf{a}\left({}^{(2)}W\right)^T + {}^{(2)}\mathbf{b}
\end{aligned}
\tag{2.4}
$$

$$
\mathbf{x} = \begin{pmatrix} x_1 & x_2 \end{pmatrix} \qquad \hat{\mathbf{q}} = \begin{pmatrix} \hat{q}_1 & \hat{q}_2 \end{pmatrix}
$$

$$
{}^{(1)}W = \begin{pmatrix} {}^{(1)}W_{11} & {}^{(1)}W_{12} \\ {}^{(1)}W_{21} & {}^{(1)}W_{22} \\ {}^{(1)}W_{31} & {}^{(1)}W_{32} \end{pmatrix} \qquad {}^{(2)}W = \begin{pmatrix} {}^{(2)}W_{11} & {}^{(2)}W_{12} & {}^{(2)}W_{13} \\ {}^{(2)}W_{21} & {}^{(2)}W_{22} & {}^{(2)}W_{23} \end{pmatrix}
$$

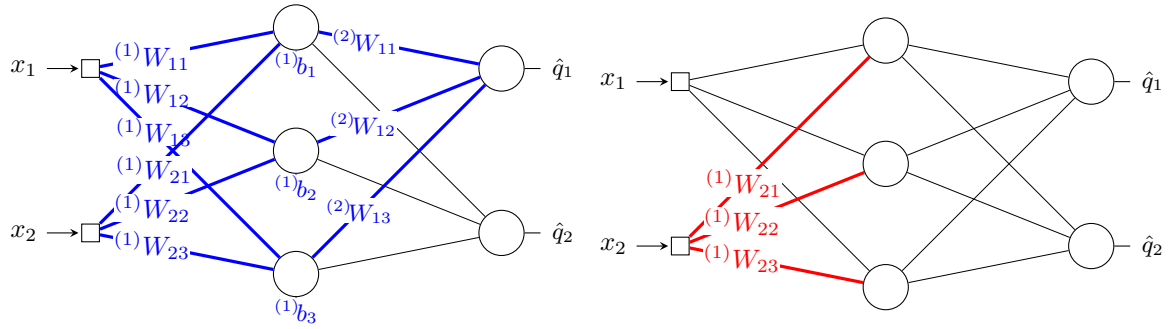
$$
{}^{(2)}\mathbf{b} = \begin{pmatrix} {}^{(1)}b_1 & {}^{(1)}b_2 & {}^{(1)}b_3 \end{pmatrix} \qquad {}^{(2)}\mathbf{b} = \begin{pmatrix} {}^{(2)}b_1 & {}^{(2)}b_2 \end{pmatrix}
$$

$$\hat{\mathbf{q}} = \sigma\left(\mathbf{x}\left({}^{(1)}W\right)^T + {}^{(1)}\mathbf{b}\right)\left({}^{(2)}W\right)^T + {}^{(2)}\mathbf{b} \tag{2.5}$$

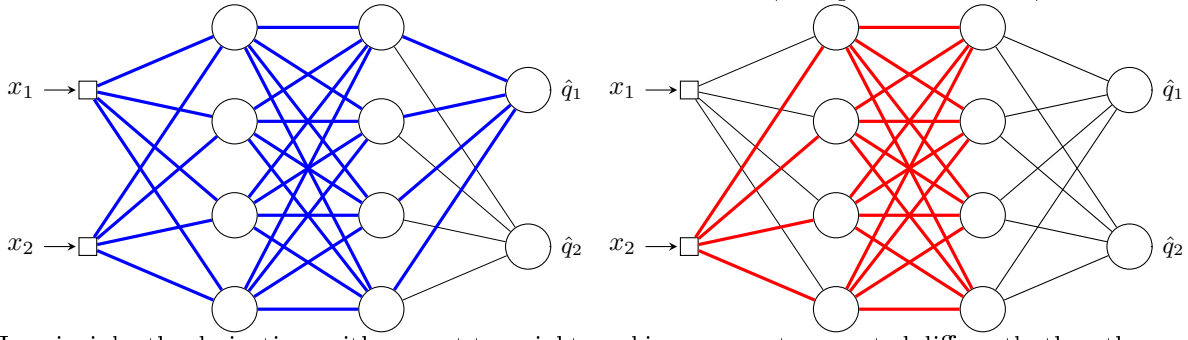
$$\begin{aligned}
\hat{q}_1 = {}&{}^{(2)}W_{11}\sigma\left({}^{(1)}W_{11}x_1 + {}^{(1)}W_{12}x_2 + {}^{(1)}b_1\right) \\
&+ {}^{(2)}W_{12}\sigma\left({}^{(1)}W_{21}x_1 + {}^{(1)}W_{22}x_2 + {}^{(1)}b_2\right) \\
&+ {}^{(2)}W_{13}\sigma\left({}^{(1)}W_{31}x_1 + {}^{(1)}W_{32}x_2 + {}^{(1)}b_3\right) \\
&+ {}^{(2)}b_1
\end{aligned}
\qquad
\begin{aligned}
\hat{q}_2 = {}&{}^{(2)}W_{21}\sigma\left({}^{(1)}W_{11}x_1 + {}^{(1)}W_{12}x_2 + {}^{(1)}b_1\right) \\
&+ {}^{(2)}W_{22}\sigma\left({}^{(1)}W_{21}x_1 + {}^{(1)}W_{22}x_2 + {}^{(1)}b_2\right) \\
&+ {}^{(2)}W_{23}\sigma\left({}^{(1)}W_{31}x_1 + {}^{(1)}W_{32}x_2 + {}^{(1)}b_3\right) \\
&+ {}^{(2)}b_2
\end{aligned}$$

If the derivative of $\hat{q}_1$ with respect to x_2 is computed, the reverse path can be followed (equation 2.6). It is also here where the gradients of the activation functions are to be used (see Figure 2.2 again). First, the inputs x_1 and x_2 are weighted and a bias is added (the weights and biases are indicated in blue), this is fed to the derivative of the activation function σ'. These outputs are weighted by the weights that are associated with the connections of all the hidden artificial neurons to the output neuron $\hat{q}_1$. However, since the chain-rule is applicable, the weights that are used to weigh the input variable x_2 within the derivative of the activation function are multiplied outside the brackets to multiply the term (indicated in red).

$$\begin{aligned}
\frac{\partial \hat{q}_1}{\partial x_2} = {}&{}^{(2)}W_{11}{}^{(1)}W_{12}\sigma'\left({}^{(1)}W_{11}x_1 + {}^{(1)}W_{12}x_2 + {}^{(1)}b_1\right) \\
&+ {}^{(2)}W_{12}{}^{(1)}W_{22}\sigma'\left({}^{(1)}W_{21}x_1 + {}^{(1)}W_{22}x_2 + {}^{(1)}b_2\right) \\
&+ {}^{(2)}W_{13}{}^{(1)}W_{32}\sigma'\left({}^{(1)}W_{31}x_1 + {}^{(1)}W_{32}x_2 + {}^{(1)}b_3\right)
\end{aligned} \tag{2.6}$$



If more layers are used, the total functions of $\hat{q}_1$ and $\hat{q}_2$ become more nested and therefore, more weighting terms are multiplied outside. Using the same logic, the forward pass is indicated in blue meaning those are used for the inputs to the derivatives of the activations and the reverse path is indicated in red, meaning those weights are terms that are multiplied to the outer function as a result of the differentiation process. The weights for the last hidden layer will remain as multiplicative terms while the bias of the output layer drops due to the differentiation (see again equation 2.6).



In principle, the derivatives with respect to weights or biases are not computed differently than the one for the inputs. The chain-rule is still applied to the nested total function for the terms that include the

particular weight. Interestingly, if the derivatives of $\hat{q}$ with respect to one of the weights that connect the last layer to the outputs, is taken. The result is just the output of the respective artificial neuron in the last layer, but not multiplied by these last weights, while the bias also drops. For the other weights closer to the input layer, again the reverse-path needs to be followed. The same applies to the biases. For the biases of the output nodes, because the output layer does not use activation functions (or linear ones, if you will), the derivative of the outputs with respect to those are just unit values. Also, note that the more artificial neurons are used, the more terms are in the full functional. The more hidden layers the artificial neural network has, the more deeper nested becomes the full functional. Both contribute to the increased expressivity of the model.

2.5 Training

Training an artificial neural network requires a loss function, that determines the 'fitness' of the artificial neural network (subsection 2.5.1). For training the artificial neural network, the backpropagation algorithm is used, which requires the gradients of the loss function (subsection 2.5.2). Lastly, the algorithms that use these gradients are explained in subsection 2.5.3 and additional considerations for artificial neural network learning in subsection 2.5.4.

2.5.1 Loss function

To train the model, a scalar variable that quantifies the fitness of the model is required. The functional that governs the performance or fit of the artificial neural network to some target data is referred to as the loss function and is already mentioned before. A typical example is data regression. Considering a data set with discrete points x with outputs q, least-squares regression with a certain set of basis functions (e.g. polynomial basis) can provide a function that approximates the data set. Similarly, an artificial neural network can represent a function or a mapping $N : x \mapsto \hat{q}$. The mapping is dependent on the artificial neural network parameters (weights and biases). By computing the difference between the output of the neural network ($\hat{q}$) and the data (q) and summing those, a scalar function can be obtained, that represents the discrepancy of the neural network to the data. There are different ways for representing the discrepancy, or the loss function, see for example [38]. The most common ones are the mean-squared error (MSE) or the sum-squared error (SSE). Less common is the discrepancy representation based on the absolute difference or so-called "Manhattan" loss function [44]. These loss functions are shown in equations 2.7.

$$\begin{aligned} \text{Mean-squared error:} \quad & MSE = \frac{1}{N_d} \sum_{j=1}^{N_o} \sum_{i=1}^{N_d} ||q_{j,i} - \hat{q}_{j,i}||^2 \\ \text{Sum-squared error:} \quad & SSE = \sum_{j=1}^{N_o} \sum_{i=1}^{N_d} ||q_{j,i} - \hat{q}_{j,i}||^2 \\ \text{Manhattan:} \quad & MAN = \sum_{j=1}^{N_o} \sum_{i=1}^{N_d} |q_{j,i} - \hat{q}_{j,i}| \end{aligned} \tag{2.7}$$

For which $\hat{q}_{j,i}$ and $q_{j,i}$ is the artificial neural network output and the target output respectively for $i = 1, \ldots, N_d$ data points and $j = 1, \ldots, N_o$ outputs of the artificial neural network. Each data point is a combination of N_i inputs and N_o outputs such that $(x_1, \ldots x_{N_i}, q_1, \ldots q_{N_o})_i$ for $i = 1, \ldots, N_d$. Each output of the artificial neural network is governed by the combination of weights and biases of the artificial neurons in the preceding layers that form the particular output. Therefore, the weights and biases have to be trained in order to minimize the loss function. Moreover, the weights and biases form the parameter space and the loss function defines the error surface [38] in that space as it is a mapping from the parameter space to a scalar function. The error surface can contain global, local minima and saddle points wherein the first one is sought for and the last two can cause difficulty when training an artificial neural network.

2.5.2 Loss gradients

As described in section 2.4, the gradients of the loss function to the weights can be computed by applying the chain rule. Let there be a artificial neural network with $\hat{q}$ being the only output. Let L be a mean-squared (MSE) loss function, for N_d data points of q. The data points are $q_i = q(x_{1_i}, x_{2_i})$ for $i = 1, \ldots, N_d$. Then for each data point, there exists an output of the artificial neural network: $\hat{q}_i$ where the inputs are: (x_{1_i}, x_{2_i}).

$$L_{MSE} = \frac{1}{N_d} \sum_{i=1}^{N_d} ||q_i - \hat{q}_i||^2 \tag{2.8}$$

As an example, the derivative of the loss function with respect to weight ${}^{(2)}W_{12}$ can be taken.

$$\frac{\partial L_{MSE}}{\partial {}^{(2)}W_{12}} = \frac{2}{N_d} \sum_{i=1}^{N_d} ||q_i - \hat{q}_i|| \left(-\frac{\partial \hat{q}_i}{\partial {}^{(2)}W_{12}} \right) \tag{2.9}$$

Which shows that the derivative of the loss function includes the difference of the artificial neural network with respect to the target output, multiplied by the derivative of the output with respect to the weight. This is computed for every data point $i = 1, \ldots, N_d$, summed and averaged by dividing by the amount of data points N_d (in this case since MSE loss-function type is used). The evaluation of $\frac{\partial \hat{q}_i}{\partial {}^{(2)}W_{12}}$ can be carried out as explained before.

2.5.3 Optimization Algorithms

Given a loss function L and a method for determining the gradients of the loss function with respect to the weights and biases. An iterative scheme for all the parameters of the artificial neural network w (weights and biases) can be formulated as [45]:

$$w_{k+1} = w_k + \alpha_k p_k \tag{2.10}$$

Where α_k is the step size and p_k is the search direction. The search direction is typically related to the gradients of the loss function with respect to the weights w. The first-order derivatives of the loss function ∇L yields a vector and the second-order derivatives $\nabla^2 L$ a matrix (called the Hessian matrix) since mixed derivatives are also possible and are situated off the diagonal. the search direction is defined as is done in [34]:

$$p_k = -B_k^{-1} \nabla L_k \tag{2.11}$$

Where B_k is a matrix that is symmetric and nonsingular. Several choices are available for B_k. The first one being simple steepest descent which chooses the identity matrix: $B_k = I$. Here, the parameters w are updated with size α_k in the direction opposite to the gradient of the loss function at the previous iteration of the parameters. The benefit of the steepest descent is that it only requires the first derivatives, however, it lacks speed on more difficult problems [34]. Different from a search direction that is steepest with regard to the slope of the loss function is the Newton-direction. Taking the Taylor-series of the loss function gives the following:

$$L(w_k + p) \approx L_k + p^T \nabla L_k + \frac{1}{2} p^T \nabla^2 L_k p \tag{2.12}$$

Taking the derivative of the Taylor series with respect to p and setting to zero gives

$$p = -(\nabla^2 L)^{-1} \nabla L \tag{2.13}$$

Which is the same as saying $B_k = \nabla^2 L$ in equation 2.11. This gives the Newton's method which is in principle equivalent to matrix inversion. The method is reliable whenever the Taylor-series in equation 2.12 is accurate, which is the case when p is small. However, inversion of the Hessian is expensive and therefore prohibitive in most cases. Therefore B_k is generally taken as the approximant of the Hessian matrix in quasi-Newton methods such as the BFGS (Broyden-Fletcher-Goldfarb-Shanno) method [12]. Here, the approximation of the Hessian is done in the following way:

$$\nabla^2 L_{k+1} (w_{k+1} - w_k) \approx \nabla L_{k+1} - \nabla L_k \tag{2.14}$$

Which gives the approximant for the Hessian matrix and which can subsequently be written as $B_{k+1}s_k = y_k$, with $s_k = w_{k+1} - w_k$ and $y_k = \nabla L_{k+1} - \nabla L_k$, this gives the following form for B_{k+1}:

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k} \tag{2.15}$$

Which is an approximate form to the Hessian matrix and which can subsequently be used in equations 2.11 and 2.10 to yield a new weight update procedure. A drawback of the BFGS method is the memory requirement, especially for larger amounts of parameters [45]. To remedy this problem, the low-memory version L-BFGS has been developed [33]. Because the optimization process often encounters local minima or saddle points, optimizers with stochasticity for machine learning purposes have been devised as well such as the Adam [24] or the AdaGrad optimizer [11].

2.5.4 Considerations

When the artificial neural network is large or the underlying function complicated, difficulty in training the artificial neural network can arise. In general, choosing the right artificial neural network and the best algorithm for learning is not straightforward ([15], [45]). Below, an overview is presented (from [5]) of important considerations and its implications when setting up an artificial neural network.

- **Data normalization:** Depending on the activation function, it is essential to normalize input data to ranges between $[-1, 1]$ or for $[0, 1]$, this prevents larger values from overriding smaller ones or it prevents premature saturation of the activation functions, which will yield derivatives being zero.
- **Network weight initialization:** At the start of the learning process, the weights and biases are initialized and assigned initial values. Typically, they are initialized uniformly in a range with zero-mean. However, if the range is too small, this could lead to very small gradients and therefore slow down the initial learning process. Occasionally, the range is defined by the number of neurons in layers, similarly as is done in [15].
- **Learning rate:** A high learning rate in gradient-descent methods will accelerate learning in the initial learning phase but could hinder convergence in later stages, where smaller step sizes are desired. Second-order methods could help remedy long training times according to [45].
- **Convergence criteria:** Longer training times do not always result in better trained artificial neural networks. Overfitting can occur, where the artificial neural network has learned patterns in the data that could be generated by noise or are non-existent. Early-stopping or proper splitting of the data set in training and test data could help against overfitting.

3

Data and Physics Informed Neural Networks

By using the spatial and temporal coordinates of the problem at hand as the inputs and the latent quantities as outputs, the artificial neural network can represent a complex mapping between the coordinates and the desired latent quantities. The complex mapping is governed by the weights and the biases of the connections and the artificial neurons respectively. The governing parameters are learned by means of a loss function that includes known data, physics as well as boundary conditions. This chapter aims to describe the framework. First on a conceptual level (Section 3.1). The reason for this is that the exact application depends on the cases considered. Next, because data-normalization is recommended practice for artificial neural networks, the scaling of the data as well as the implications on the physics incorporation is discussed in Section 3.2. This concludes the general discussion on the framework. lastly, the implementation of the framework into `Python` is discussed in Section 3.3.

3.1 Methodology

For this section, the general formulation is given, which is based on and inspired by the formulations given in [28]. In general, let $\mathbf{s}$ be a vector in $\mathbb{R}^n$. For $\Omega \subset \mathbb{R}^n$ a functional exists $\mathbf{q} = \mathbf{q}(s) : \mathbb{R}^n \rightarrow \mathbb{R}^k$. The functional $\mathbf{q}$ satisfies a partial differential equation with a general formulation given by (3.1) within the domain of $\mathbf{s} \in \Omega$. On the boundary of Ω, $\partial\Omega$, boundary conditions for $\mathbf{q}$ are known that close the problem.

$$f\left(s_1, \dots s_n; \mathbf{q}, \frac{\partial \mathbf{q}}{\partial s_1}, \dots \frac{\partial \mathbf{q}}{\partial s_n}; \frac{\partial^2 \mathbf{q}}{\partial s_1 \partial s_1}, \cdots \frac{\partial^2 \mathbf{q}}{\partial s_1 \partial s_n}; \dots\right) = 0 \tag{3.1}$$

Thus, by solving the partial differential equation in conjunction with the boundary conditions, the functional $\mathbf{q}(\mathbf{s})$ can be found. This is a general formulation, but for physical problems, the $\mathbf{s}$ can be a collection of space and time variables: $\mathbf{s} = (t, \mathbf{x})$. For example, $\mathbf{q}$ could be a passive scalar ϕ e.g. when f is the passive scalar equation. Or, $\mathbf{q}$ can be velocity and pressure: $\mathbf{q} = (\mathbf{u}, p)$, when the Navier-Stokes equations are considered for f. Furthermore, f could either be a linear or non-linear functional. For some problems, analytical solutions exist, while for others numerical tools are employed to approximate the solution. Here, a different approach is considered. One that uses the approximation power of artificial neural networks to represent the (complex) solution. By taking $\mathbf{s}$ as the inputs for an artificial neural network, while taking $\mathbf{q}$ as outputs, a complex functional can be obtained by using the artificial neural network. The outputs of the artificial neural network will be denoted by hats over them, thus for the general problem: $\hat{\mathbf{q}}$ are the artificial neural network outputs, which will be the approximant to $\mathbf{q}$.

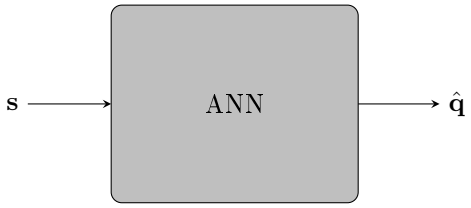


Figure 3.1: General artificial neural network for DPINN

The goal is to train the artificial neural network such that $\hat{\mathbf{q}} \approx \mathbf{q}$. Artificial neural networks are trained by means of the backpropagation algorithm (which utilize the optimizer algorithms described in 2.5.3), by using the same algorithm, the derivatives of the outputs with respect to the inputs can be computed. These derivatives can then be used to form the governing (physical) equations. If next to the governing equations, as well as the boundary conditions, some limited knowledge on $\mathbf{q}$ is available in Ω, a training target for the artificial neural networks such that it approximates the true latent function, can be formulated. This target is referred to as the loss function. For this approach, the governing equations, the available data and possibly the boundary conditions are included in the loss function equation 3.2. Hence the approach is referred to as data and physics informed neural networks (DPINN).

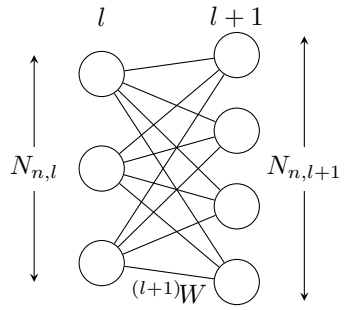
$$L = L_d + L_f(+L_{bc}) \tag{3.2}$$

The outputs, as well as the derivatives, are dictated by the complexity and the architecture of the artificial neural networks. If the model is complex enough, the artificial neural network can represent the solution to the problem given. In the following, the model, as well as the data and physics incorporation, is explained. Subsequently, the optimization strategy is discussed.

3.1.1 Model

The artificial neural network model is static and pre-defined before the training phase for a problem at hand. Choosing the size/complexity of the artificial neural network requires taking into account two things. The first is to choose an artificial neural network that is complex enough i.e. that it contains enough hidden layers and artificial neurons. An artificial neural network with only one hidden-layer will most likely not be sufficient for problems that contain lots of features, as the first layer is mostly for subdividing the input space. The more artificial neurons the first layer contains, the more subdivisions

are possible. Because of this, the output of the first layer is increased in dimension which can then be fed into the next hidden layer and so forth. Increasing the number of hidden layers and/or artificial neurons in them increases the expressivity of the artificial neural network. However, it could suffer from decreased trainability, since more complex artificial neurons are harder to train. Since trainability is also largely determined by the initialization of the artificial neural network. When setting up the network, an initial set of weights and biases need to be chosen. Important here is that if these sets are chosen too large in absolute sense, the responses of the neurons are towards the high and low ends of the derivatives activation functions and the derivatives of the Loss function with respect to the weights and biases can become large or very small. Which could not be beneficial for the optimization scheme. In general, the biases are initialized as zeros and the weights are initialized by means of Xavier initialization [15], where the weights in each layers are initialized by sampling a truncated normal distribution with a standard deviation equal to $\beta = \sqrt{\frac{2}{N_{n,l-1}+N_{n,l+1}}}$. So, for the weights between the layers l and $l+1$, they are sampled from a normal distribution with mean zero and the standard deviation β. If the drawn weight is larger than 2β in magnitude, the weight is redrawn, hence the truncated normal distribution.



$$\beta = \sqrt{\frac{2}{N_{n,l-1} + N_{n,l+1}}}$$

$${}^{(l+1)}W_{i,j} \sim \mathcal{N}\left(0, \beta^2\right)$$

$${}^{(l+1)}W = \begin{pmatrix} {}^{(l+1)}W_{11} & {}^{(l+1)}W_{12} & \cdots \\ \vdots & \ddots & \\ {}^{(l+1)}W_{N_{n,l+1}1} & & {}^{(l+1)}W_{N_{n,l+1}N_{n,l}} \end{pmatrix}$$

3.1.2 Data informing

By informing the artificial neural network, it is meant that the loss function for training is augmented with the information known. Which could be information in the form of discrete datapoints in Ω. For these datapoints, some or all of the variables in the solution $\mathbf{q}$ are known. The datapoints are collected in a set $\mathbf{S}_d = (\mathbf{s}_{d,1}, \ldots, \mathbf{s}_{d,N_d}) \in \mathbb{R}^{N_d \times n}$, $\mathbf{s}_{d,i} \in \Omega \subset \mathbb{R}^n$. Whereas the variables that are known are included in the set $\mathbf{D} = (\mathbf{d}_1, \ldots, \mathbf{d}_{N_d}) \in \mathbb{R}^{N_d \times d}$. Where $\mathbf{d}_i \in \mathbb{R}^{1 \times d}$, $d \leq k$, a vector that contains the solution for some or all of the variables in $\mathbf{q}(\mathbf{s}_{d,i})$. In order to include the data into the Loss function, a measure for the discrepancy needs to be computed. By inserting each data point $\mathbf{s}_{d,i}$ of $\mathbf{S}_d$ in the artificial neural network, $\hat{\mathbf{q}}_{d,i}$ is obtained i.e. the predicted approximate solution of the model for each datapoint. This form the set $\hat{\mathbf{Q}}_d$, which contains all the outputs of the model for each data point. By taking the same type of variables - for which information is known - from it, the set $\hat{\mathbf{D}}$ is obtained. For example, when velocity $\mathbf{u}$ is known on these points and when $\hat{\mathbf{q}} = (\mathbf{u}, p)$, the the velocity data is in $\mathbf{D}$. The discrepancy between the set of known solutions $\mathbf{D}$ and $\hat{\mathbf{D}}$ for $i = 1, \ldots N_d$ can then be computed to be used in the data loss function:

$$L_d = L_d(\mathbf{D}, \hat{\mathbf{D}}) \tag{3.3}$$

Where L_d is a function that computes the discrepancy between all the solutions in the set and computes the scalar loss function that represents the error between the true and approximate solution. L_d can e.g. be either the MSE or SSE loss function (see equation 2.7 in subsection 2.5.1).

3.1.3 Physics informing

Next to the discrete data information on $\mathbf{q}$, the governing equations for $\mathbf{q}$ are known and can therefore be provided as information to the framework. It is assumed that the artificial neural network is complex enough that $\hat{\mathbf{q}} \approx \mathbf{q}$ and therefore the governing equations are valid for $\hat{\mathbf{q}}$ as well:

$$f\left(s_1, \ldots s_n; \hat{\mathbf{q}}, \frac{\partial \hat{\mathbf{q}}}{\partial s_1}, \ldots \frac{\partial \hat{\mathbf{q}}}{\partial s_n}; \frac{\partial^2 \hat{\mathbf{q}}}{\partial s_1 \partial s_1}, \cdots \frac{\partial^2 \hat{\mathbf{q}}}{\partial s_1 \partial s_n}; \ldots\right) = 0 \tag{3.4}$$

These governing equations can be readily computed, all it needs is a method of evaluating the governing equations throughout the domain Ω. For this, a similar least-squares approach is used. For this, the domain is sampled by physics collocation points, which results in the set $\mathbf{S}_f = \left(\mathbf{s}_{f,1}, \ldots, \mathbf{s}_{f,N_f}\right)$. Within the set, the datapoints set $\mathbf{S}_d$ can be included, to try to minimize the physics error on the datapoints as well. For each point in $\mathbf{S}_f$, the physics can be computed which results in the set $\mathbf{F}$. Ideally, $\mathbf{F} = 0$ which means that the artificial neural network produces a solution that is perfectly compliant to the governing equations or it is the trivial solution (all variables constant such that the gradients and the governing equations are too). But it is assumed that the combination of the data and physics prevents the artificial neural network from converging to the trivial solution. In practice, $\mathbf{F} = 0$ a target for minimization, therefore, needs to be included in the loss function, which is done in physics loss function:

$$L_f = L_f(\mathbf{F}) \tag{3.5}$$

For which again, L_f is a function that computes a scalar value from $\mathbf{F}$, to be able to be included in the total loss function. This function can again be the MSE or the SSE loss function (equation 2.7)

3.1.4 Boundary conditions informing

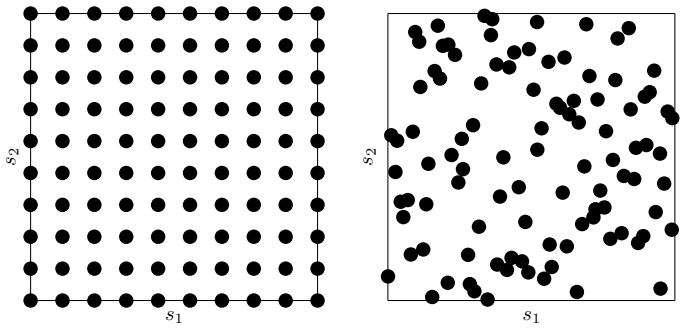
Furthermore, the region Ω is assumed to be bounded by $\partial\Omega$, on which boundary conditions for $\mathbf{q}$ are known. The boundary conditions can be of Dirichlet type. Let $b(\mathbf{q})$ be the boundary conditions for $\mathbf{q}$ on $\partial\Omega$ then similarly, the discrepancy between $b(\mathbf{q})$ and $b(\hat{\mathbf{q}})$ can be incorporated in the loss function for boundary conditions (again with MSE or SSE types) :

$$L_{bc} = L_{bc}(b(\mathbf{q}), b(\hat{\mathbf{q}})) \tag{3.6}$$

If the boundary conditions are of a Dirichlet type, the loss function for these boundary conditions is in practice no different than the one used for the data loss function, in the sense that the solutions are known on some or all boundaries in $\partial\Omega$ and it is merely data informing. By sampling these boundaries with discrete points, the discrepancy of the artificial neural network on these boundaries can be computed similarly. For Neumann or Robin boundary conditions, the gradient needs to be computed, and is therefore practically not very different than the loss function for the physics in the sense that gradients of the approximate solution are used. Nevertheless, for completeness, the boundary conditions are discussed here. The sampling of the boundaries results in the set of points $\mathbf{S}_{bc}$. If $\partial\Omega$ contains boundaries with Dirichlet boundary conditions, a set of solution values $\mathbf{Q}_{bc}$ is required.

3.1.5 Sampling

Because of the least-squares approach, the loss functions are computed on sets of discrete points $\mathbf{S}_d$, $\mathbf{S}_f$ and $\mathbf{S}_{bc}$ for data, physics and boundary conditions respectively. A strategy for sampling Ω and $\partial\Omega$ and generating these sets of discrete is required. For the remainder of this work, the domains are rectangular-shaped but this does not necessarily have to be the case, any shape can be discrete sampled (see e.g. [9] or [29]). Regardless, different methods for sampling can be used e.g. uniformly or randomly (Figure 3.2)



(a) Uniform (b) Random

Figure 3.2: Domain sampling techniques

A continuous-field sampling technique for data can not always be chosen and can be predefined by the way the data is obtained. For example, if it is by experiments or from simulations, the same applies to the number of data points N_d, which is often fixed. This is not the case for the physics collocation points N_f. If the domain is known, physics collocation samples can be sampled anywhere. An example of sets of points is plotted in Figure 3.3. Here, the data points are uniformly distributed, while the physics collocation points, as well as the boundary points, are randomly distributed. For the remainder of this document, the set of points for data $\mathbf{S}_d$ will be plotted by blue squares always, while $\mathbf{S}_f$ will be red circles and boundary points $\mathbf{S}_{bc}$ black crosses, for distinguishing purposes.

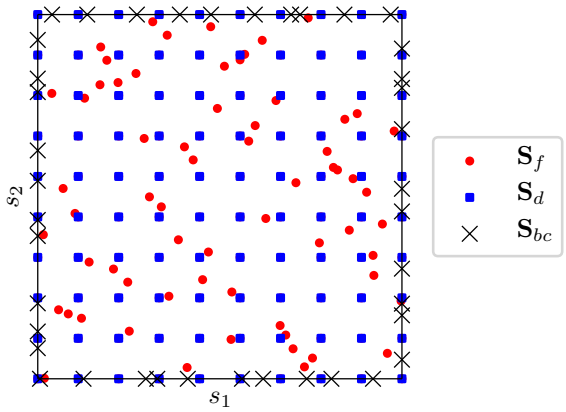


Figure 3.3: Domain sampling

3.1.6 Optimization/training

For training the artificial neural network, the loss function is set-up in the way explained previously. Where the domain is sampled and all the datapoints are available. Next, the optimization strategy has to be chosen. There are several types of optimizers available but the choice here is to use a combination of two types of optimizers. This strategy is suggested by [9]. The optimizers used are the Adam optimizer and the L-BFGS optimizer (see 2.5.3). The latter optimizer is the main optimizer whereas the Adam optimizer is an optimizer that is mainly used to augment the training process. For example, when the L-BGSB optimizer line search fails, the stochastic character of the Adam optimizer is run for about 1000 iterations, with the idea to help the L-BFGS out of the troublesome region [9], if one is encountered.

3.1.7 Predict

Once the artificial neural network training process has finished. The domain Ω is uniformly sampled with a large number of points, for which the outputs and all the relevant gradients are evaluated. By using a large number of samples, the overall solution can be examined. Since the gradients are available through automatic differentiation, the governing equations can be computed, which are then able to give an indication of the error in the physics that is made within the domain. It is customary/required within machine learning, to test the models on inputs that the model has not seen in the training process. By sampling the domain with a number of points that is much larger than all the datapoints used in the training process, it is assumed that this gives enough unseen inputs, for this requirement to be satisfied.

3.2 Scaling

A particular requirement for inputs and outputs of the artificial neural network is that they are close to a range $[-1, 1]$ for each input (the $[-1, 1]$ will be referred to as the unit range or NN range where NN stands for Neural Network), as the learning process benefits from this. However, $\mathbf{s}$ nor $\mathbf{q}$ are not necessarily in that range but are in Ω and $\mathcal{Q}$ respectively. By using a length scale δ_s present in Ω as well as reference values $\mathbf{q}_0$, $\mathbf{s}$ and $\mathbf{q}$ can be transferred to their non-dimensional counterparts: $\mathbf{s}^*$ and $\mathbf{q}^*$ (equations 3.7 and 3.10), which are in Ω^* and $\mathcal{Q}^*$ respectively. Note that, the superscript asterisk $(*)$ denotes that the quantities are in non-dimensional form. However, the length scale does not guarantee that the quantities involved are close to the unit range. An additional step can be carried out that transfers the inputs to the unit range. This can be done by using the minima and maxima of $\mathbf{s}^* \in \Omega^*$ to form a non-dimensional length scale δ_s^* as in equation 3.8, where g is a function that takes the maximum over the resulting vector. Subsequently, the inputs can be scaled as in equation 3.9 . This will transform the inputs $\mathbf{s}^*$ to $\mathbf{s}^{\hat{*}}$ which is close to the unit range $\Omega^{\hat{*}}$.

$$\mathbf{s}^* = \frac{\mathbf{s}}{\delta_s} \quad (3.7) \qquad \delta_s^* = g(\mathbf{s}^*_{max} - \mathbf{s}^*_{min}) \quad (3.8) \qquad \mathbf{s}^{\hat{*}} = \frac{\mathbf{s}^*}{\delta_s^*} \quad (3.9)$$

For the quantities $\mathbf{q}^*$, care has to be taken to not scale individual quantities differently, as this would also affect terms in the governing equations of f, not rendering them zero anymore. Moreover, it's not necessarily the case that there is information on the magnitudes of all quantities in $\mathbf{q}^*$. Nevertheless, a scaling factor can be obtained from the quantities that are known. Let $\mathbf{D}^*$ again be the set of data quantities, but this time, they are scaled by their respective reference values $\mathbf{q}_0$. If the data set is large enough, an estimate on the magnitudes for all variables in $\mathbf{D}^*$ can be obtained. Bear in mind, that $\mathbf{D}^*$ is a subset of $\mathbf{Q}^*$, all the outputs of the artificial neural network for a particular set of inputs. By computing the norm of $\mathbf{D}^*$, a row vector $||\mathbf{D}^*|| \in \mathbb{R}^{N_d \times 1}$ is obtained (equation 3.12) .

$$\mathbf{q}^* = \frac{\mathbf{q}}{\mathbf{q}_0} \quad (3.10) \qquad \mathbf{q}^{\hat{*}} = \frac{\mathbf{q}^*}{S_c^*} \quad (3.11)$$

$$||\mathbf{D}^*|| = \begin{pmatrix} \sqrt{{d^*_{1,1}}^2 + \ldots + {d^*_{d,1}}^2} \\ \cdots \\ \sqrt{{d^*_{1,N_d}}^2 + \ldots + {d^*_{d,N_d}}^2} \end{pmatrix} \quad (3.12) \qquad S_c^* = ||\mathbf{D}^*||_{max} - ||\mathbf{D}^*||_{min} \quad (3.13)$$

If now the maxima and minima of these are taken, a scaling factor S_c^* can be obtained as is done in equation 3.13. The reason for the two-step transformation is because the governing equations utilized for the inclusion of physics can either be in dimensional-form or in non-dimensional form. Moreover, a consistent transformation between Ω, Ω^* and $\Omega^{\hat{*}}$ is required. In principle, transforming $\mathbf{q}^*$ to $\mathbf{q}^{\hat{*}}$ requires information on the magnitude for each component of $\mathbf{q}$, which might not be readily available if one or more components of $\mathbf{q}$ are to be inferred by means of data assimilation. Therefore, the scaling S_c^* is applied to these to be inferred variables depending on the governing equations used.
Nevertheless, because the data normalization is recommended practice for artificial neural networks, the inputs and outputs of the artificial neural network will be taken as $\hat{\mathbf{s}}^{\hat{*}}$ and $\hat{\mathbf{q}}^{\hat{*}}$ respectively. the same scaling factor S_c^* is applied to all variables, as long as it is done in a consistent fashion. While the inputs and outputs are always taken as $\hat{\mathbf{s}}^{\hat{*}}$ and $\hat{\mathbf{q}}^{\hat{*}}$ respectively, the computation of the physics can either be computed in one of the three forms:

- In dimensional form (D-form):
  $f\left(s_1, \ldots s_n; \hat{\mathbf{q}}, \frac{\partial \hat{\mathbf{q}}}{\partial s_1}, \ldots \frac{\partial \hat{\mathbf{q}}}{\partial s_n}; \frac{\partial^2 \hat{\mathbf{q}}}{\partial s_1 \partial s_1}, \cdots \frac{\partial^2 \hat{\mathbf{q}}}{\partial s_1 \partial s_n}; \ldots\right) = 0$
- In non-dimensional form (ND-form):
  $f^*\left(s_1^*, \ldots s_n^*; \hat{\mathbf{q}}^*, \frac{\partial \hat{\mathbf{q}}^*}{\partial s_1^*}, \ldots \frac{\partial \hat{\mathbf{q}}^*}{\partial s_n^*}; \frac{\partial^2 \hat{\mathbf{q}}^*}{\partial s_1^* \partial s_1^*}, \cdots \frac{\partial^2 \hat{\mathbf{q}}^*}{\partial s_1^* \partial s_n^*}; \ldots\right) = 0$
- In (scaled) unit-range form (NN-form):
  $f^{\hat{*}}\left(s_1^{\hat{*}}, \ldots s_n^{\hat{*}}; \hat{\mathbf{q}}^{\hat{*}}, \frac{\partial \hat{\mathbf{q}}^{\hat{*}}}{\partial s_1^{\hat{*}}}, \ldots \frac{\partial \hat{\mathbf{q}}^{\hat{*}}}{\partial s_n^{\hat{*}}}; \frac{\partial^2 \hat{\mathbf{q}}^{\hat{*}}}{\partial s_1^{\hat{*}} \partial s_1^{\hat{*}}}, \cdots \frac{\partial^2 \hat{\mathbf{q}}^{\hat{*}}}{\partial s_1^{\hat{*}} \partial s_n^{\hat{*}}}; \ldots\right) = 0$

If the physics equations are analyzed in the unit-range form, no additional steps are required as the inputs and outputs are already in unit-range form and the gradients will already be as well. However, when the other forms are used, the gradients and variables need to be scaled inversely. For example, if the derivative of $\hat{q}_i$ with respect to $\hat{s}_j$ (the i-th output with respect to the j-th input) is required to compute the physics in dimensional-form, this can be done by:

$$\begin{aligned}\frac{\partial \hat{q}_i}{\partial \hat{s}_j} &= \frac{\partial \hat{s}_j^*}{\partial \hat{s}_j}\frac{\partial \hat{q}_i}{\partial \hat{q}_i^*}\frac{\partial \hat{q}_i^*}{\partial \hat{s}_j^*} \\ &= \left(\frac{\partial \hat{s}_j^*}{\partial \hat{s}_j}\frac{\partial \hat{s}_j^{\hat{*}}}{\partial \hat{s}_j^*}\right)\left(\frac{\partial \hat{q}_i}{\partial \hat{q}_i^*}\frac{\partial \hat{q}_i^*}{\partial \hat{q}_i^{\hat{*}}}\right)\frac{\partial \hat{q}_i^{\hat{*}}}{\partial \hat{s}_j^{\hat{*}}}\end{aligned}$$

Which is the application of the chain-rule and where the gradient $\frac{\partial \hat{\mathbf{q}}_i^{\hat{*}}}{\partial \hat{s}_j^{\hat{*}}}$ is obtained by the framework. The multiplicative terms can be computed by the transformation introduced before for example by:

$$\frac{\partial \hat{s}_j^{\hat{*}}}{\partial \hat{s}_j^*} = \frac{\partial}{\partial \hat{s}_j}\left(\frac{\hat{s}_j^*}{\delta_s^*}\right) = \frac{1}{\delta_s^*} \qquad \frac{\partial \hat{s}_j^*}{\partial \hat{s}_j} = \frac{\partial}{\partial \hat{s}_j}\left(\frac{\hat{s}_j}{\delta_s}\right) = \frac{1}{\delta_s}$$

$$\frac{\partial \hat{q}_i}{\partial q_i^*} = \frac{\partial}{\partial \hat{q}_i^*}\left(\hat{q}_i^* q_{0,i}\right) = q_{0,i} \qquad \frac{\partial \hat{q}_i^*}{\partial \hat{q}_i^{\hat{*}}} = \frac{\partial}{\partial \hat{q}_i^{\hat{*}}}\left(\hat{q}_i^{\hat{*}} S_c^*\right) = S_c^*$$

$$\begin{aligned}\frac{\partial \hat{q}_i}{\partial \hat{s}_j} &= \left(\frac{\partial \hat{s}_j^*}{\partial \hat{s}_j}\frac{\partial \hat{s}_j^{\hat{*}}}{\partial \hat{s}_j^*}\right)\left(\frac{\partial \hat{q}_i}{\partial \hat{q}_i^*}\frac{\partial \hat{q}_i^*}{\partial \hat{q}_i^{\hat{*}}}\right)\frac{\partial \hat{q}_i^{\hat{*}}}{\partial \hat{s}_j^{\hat{*}}} \\ &= \left(\frac{1}{\delta_s}\frac{1}{\delta_s^*}\right)\left(q_{0,i}S_c^*\right)\frac{\partial \hat{q}_i^{\hat{*}}}{\partial \hat{s}_j^{\hat{*}}}\end{aligned}$$

For the loss functions, this means that the data loss function will use the data values that are in the unit-range form, rather than the dimensional form and thus the original functional for the data loss function (equation 3.3) is adjusted to:

$$Loss_d = L_d(\mathbf{D}^{\hat{*}}, \hat{\mathbf{D}}^{\hat{*}}) \tag{3.14}$$

While for the physics loss function, as mentioned, the choice can be made to either use dimensional, non-dimensional or unit range forms of the governing equations, thus the options for equation 3.5 are:

$$\begin{aligned}Loss_f &= L_f(\mathbf{F}) \\ Loss_f &= L_f(\mathbf{F}^*) \\ Loss_f &= L_f(\mathbf{F}^{\hat{*}})\end{aligned} \tag{3.15}$$

Throughout the remainder of this chapter, no distinction is made between the forms, however, the inputs and outputs of the model are always in theunit-range form ($\hat{\mathbf{s}}^{\hat{*}}$ and $\hat{\mathbf{q}}^{\hat{*}}$ respectively), the superscripts are omitted for readability purposes.

3.3 Implementation

This section is devoted to the coding and implementation of the DPINN framework, which is described on a high level in the previous section. The framework is subdivided into modules, which are coded in the `Python` programming language. Within `Python`, third-party packages can be used to aid the different coding routines. The main third-party packages used for the framework are `NumPy` (`np`), `TensorFlow` (`tf`) and `Matplotlib`. Here, the first one is for handling arrays and numerical computations, `TensorFlow` is built for creating artificial neural networks and training them [1]. More specifically `TensorFlow` provides high-level coding for machine learning with GPU-learning capabilities. Lastly, `Matplotlib` provides routines for plotting purposes. Moreover, on `TensorFlow`, it is a symbolic math library developed by Google Brain Team. It is specifically focused on machine learning applications. As a package in Python (3.6), `TensorFlow` provides a high-level library for graph creation and manipulation with CUDA capabilities. Which allows for computations on GPU's (in this case the NVIDIA Titan V GPU, kindly provided by NVIDIA). The creation of computational graphs is done by making manipulations (such as products, additions, etc.) to special `TensorFlow` placeholders and variables. The outputs are then undefined `TensorFlow` `tf.Tensors`. Once variables of the graph are initialized and specified, the actual evaluation of the graphs is carried out by a `TensorFlow` `tf.session`. This is done by passing values or arrays to a python dictionary that points to the input `TensorFlow` placeholders `tf.placeholder`. This dictionary can then be passed to a `TensorFlow` `tf.session` to evaluate the model and generate numerical outputs. For example, if the model is created by taking `tf.placeholder` $\mathbf{s}_{\texttt{tf}}$ as inputs and $\hat{\mathbf{q}}_{\texttt{tf}}$ as outputs, it can be evaluated by feeding `np.array` $\mathbf{s}$ to a `dictionary` for $\mathbf{s}_{\texttt{tf}}$, a numerical output $\hat{\mathbf{q}}$ can be obtained by passing it to a `tf.session` and running it (Figure 3.4). In this section, the asterisks and hats are omitted on the inputs and outputs, for clarity purposes but it should be noted that these inputs and outputs are always in the unit range form.

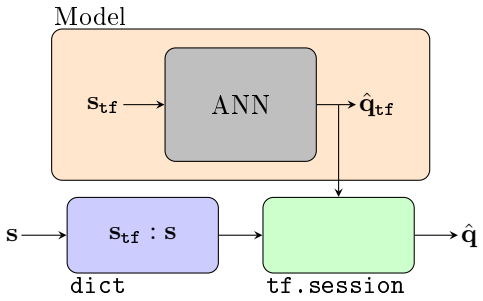


Figure 3.4: `TensorFlow` framework session passing schematic for vectors

3.3.1 Model

In this framework, a model is referred to as the artificial neural network, with the parameters: (N_i, N_n, N_o, La, σ) including the inputs and outputs. The inputs N_i are equal to the dimension of $\mathbf{s}$: $N_i = n$. Furthermore, the outputs N_o are equal to the dimension of $\mathbf{q}$: $N_o = k$. An important feature of the `tf.placeholder` is that they can be dynamically sized, so sets of inputs can also be passed to the model without knowing the number of inputs (Figure 3.5). The inputs are taken as column vectors, therefore adding extra points to a set $\mathbf{S}$ is just adding rows to an `np.array`.

The artificial neural network is set up once and will be used for all the computational but with different placeholders as inputs and therefore also different outputs. The artificial neural network has a set of weights as well as a set of biases. By using `tf.matmul` for tensor products, `tf.add` for additions and predefined function such as the `tf.nn.tanh` for the activation functions σ, the artificial neural network can be formed which is shown in Figure 3.6.

3.3.2 Data

Data points for training the artificial neural networks can either be generated by analytical solutions of cases, by using CFD solutions or by experimental solutions. Regardless, an array with a set of coordinates $\mathbf{S}_d = (\mathbf{s}_{d,1}, \ldots, \mathbf{s}_{d,N_d}) \in \mathbb{R}^{N_d \times N_{in}}$ is required. For each coordinate, data on one or multiple variables of $\mathbf{q}$ is known. Therefore, next to $\mathbf{S}_d$, `np.array` $\mathbf{D}$ exists. For example, if the framework is used for pressure inference from 2D velocity data, $\mathbf{D} = ((u_d, v_d)_1, \ldots, (u_d, v_d)_{N_d})$. The data-points

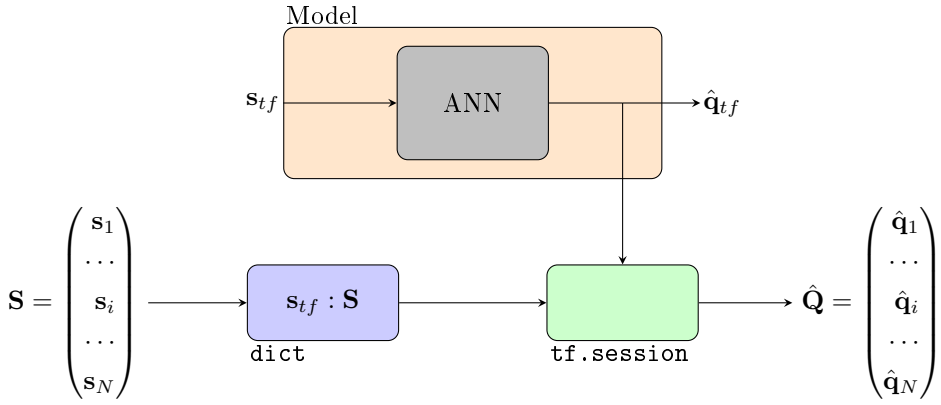


Figure 3.5: `TensorFlow` framework session passing schematic for multidimensional vectors

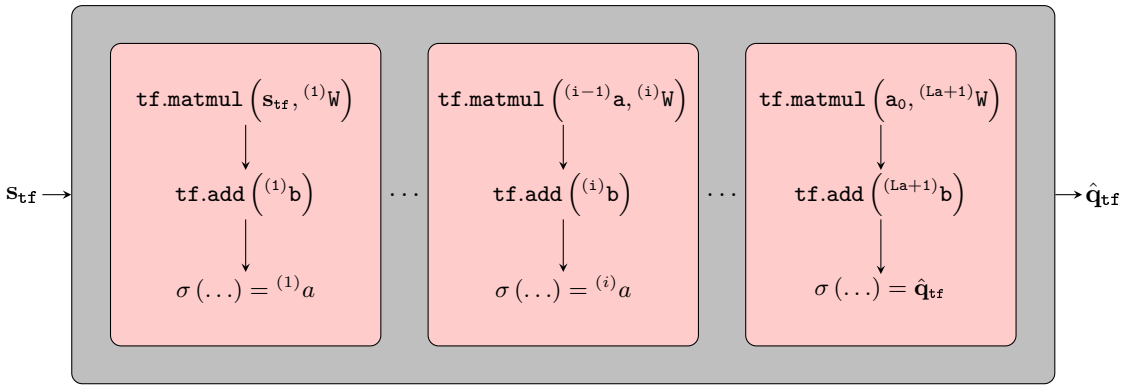


Figure 3.6: Construction of the artificial neural network in `TensorFlow`

$\mathbf{S}_d$ are fed to the graph by means of a Python dictionary, the outputs are then $\hat{\mathbf{Q}}_d$, which contain the outputs $\hat{\mathbf{D}}$. Both $\mathbf{D}$ and $\hat{\mathbf{D}}$ can then be used to evaluate the data loss function (equation 3.3)

3.3.3 Gradients

In theory, the derivatives of the artificial neural network can be computed by symbolic differentiation, numerical differentiation or automatic differentiation. Symbolic differentiation is the automatic manipulation of functional expressions whereas numerical differentiation is the finite difference approximation of the derivatives. Automatic differentiation is neither of those. A function, which is to be differentiated, can be seen as a series or nested structure of elementary operations for which the derivatives are known. [6] gives an example for the function $f(x, y) = ln(x_1) + x_1x_2 - \sin(x_2)$, for which the computational graph is shown in Figure 3.7. Where v_{-1}, v_0 are input variables, v_5 is the output variable and the others are intermediate variables (another example can be found in [34]).

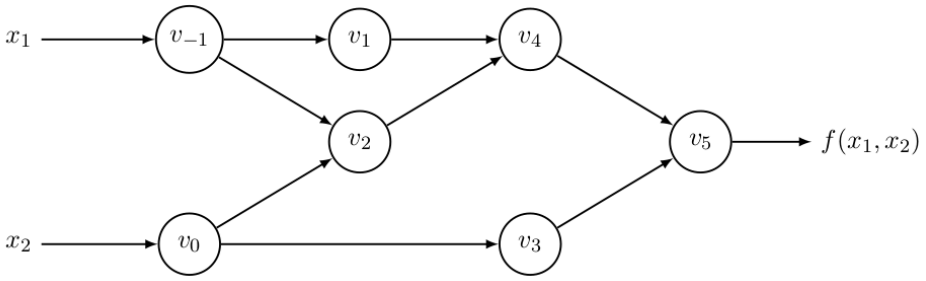


Figure 3.7: Computational graph for $f(x, y) = ln(x_1) + x_1x_2 - \sin(x_2)$, from [6]

In order to compute the derivative of f with respect to an input e.g. x_1, for every (intermediate) variable a derivative with respect to the input can be computed: $\dot{v}_i = \frac{\partial v_i}{\partial x_1}$. By following the computational graph, the derivative of f with respect to x_1 can then be found. This is referred to as forward-mode

AD. Much less costly is the reverse-mode AD. Here, the inputs are fed through the graph to populate the intermediate values, then by propagating adjoints from the outputs to the inputs backwards, the derivative of f with respect to x_1 can then also be found. Because an artificial neural network model can also be seen as a complex graph with elementary operations, automatic differentiation is well suited for the task. In fact, the `TensorFlow` package comes with automatic differentiation capabilities, which are embedded in the function `tf.gradients`.

3.3.4 Physics

By using the `tf.gradients` functionality, the model can be differentiated with respect to the inputs ($\mathbf{s}_{\mathtt{tf}}$) by using `tf.gradients`. By combining the gradients and the outputs the governing equations can be constructed. The governing equations are still dependent on the inputs and therefore, an array with a set of coordinates $\mathbf{S}_f = (\mathbf{s}_{f,1}, \ldots, \mathbf{s}_{f,N_f}) \in \mathbb{R}^{N_f \times N_{in}}$ is required. These points will then be used to generate $\mathbf{F}$, which can be used in the physics loss function. Depending on the number of governing equations, the physics loss function can contain multiple terms.

3.3.5 Boundary Conditions

As an example, let the boundary $\partial\Omega$ be divided into two boundaries: $\partial\Omega_{Dir}$, $\partial\Omega_{Neum}$, where on the first one Dirichlet boundary conditions for $q_1^{\hat{*}}$ are employed, being $q_1 = q_{1,Dir}$. Secondly, let the second boundary have a Neumann boundary condition for $q_2^{\hat{*}}$: $\frac{\partial q_1}{\partial s_2} = 0$. Then for both, the boundaries are sampled which results into two sets of datapoints $\mathbf{S}_{BC,Dir} \in \mathbb{R}^{N_{BC} \times N_{in}}$ and $\mathbf{S}_{BC,Neum} \in \mathbb{R}^{N_{BC} \times N_{in}}$. If both sets are fed to the artificial neural network model, the results is two sets of outputs: $\hat{\mathbf{Q}}_{BC,Dir}$ and $\hat{\mathbf{Q}}_{BC,Neum}$. The first is the output for the Dirichlet boundary condition, which can readily be compared to the desired boundary condition $q_{1,Dir}$. Whereas for the Neumann boundary condition, the gradient of q_1 needs to be computed first with respect to s_2, this can be carried out by `tf.gradients`, just as for the physics.

3.3.6 Loss function

By comparing $\hat{\mathbf{Q}}_d$ to $\mathbf{Q}_d$, the discrepancy between the model and the target outputs can be obtained, for the data-points given. The discrepancy can be expressed by the MSE or SSE. `TensorFlow` provides routines for minimizing means as well as sums (`tf.reduce_mean` and `tf.reduce_sum`), while the squares can be computed as normally done on `np` arrays. The result is a scalar value in the form of a `tf.Tensor`. Evaluating the loss function can be done again by passing arrays for the data-points to a dictionary and feeding it as well as the $Loss_{d_{\mathtt{tf}}}$ to a `tf.session` (Figure 3.8). If only data is available for some of the variables, it is just a matter of selecting the right variables from the outputs.

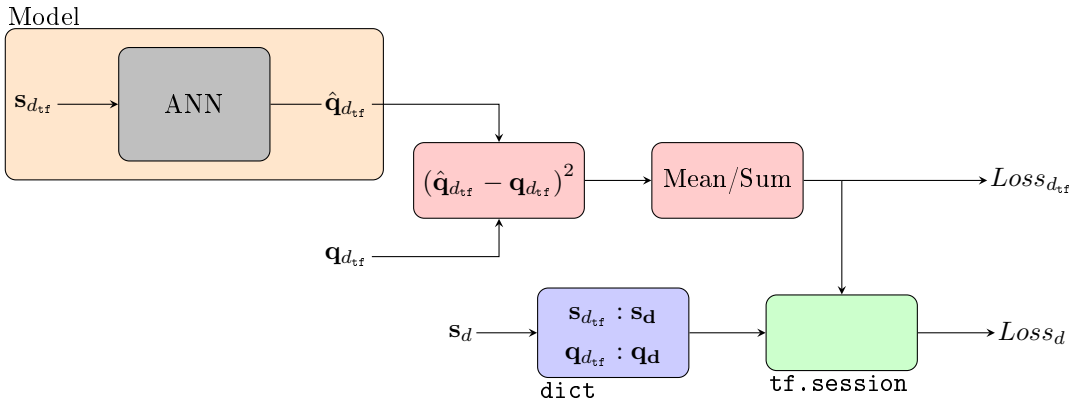


Figure 3.8: Construction of the data loss function in `TensorFlow`

By passing $\mathbf{s}_f$ to the model, the output of the governing equation for the passed points can be obtained. Similarly as done for the data loss function, the physics loss function can be obtained by taking the mean (MSE) or sum (SSE) squared error of the output $f_{\mathtt{tf}}$ (see Figure 3.9).
Again, for the Dirichlet boundary conditions, there is practically no distinction in the implementation in comparison to the data loss function. The only difference is that the datapoints are situated on the boundaries. For the Neumann boundary conditions, the gradients are calculated for the given points

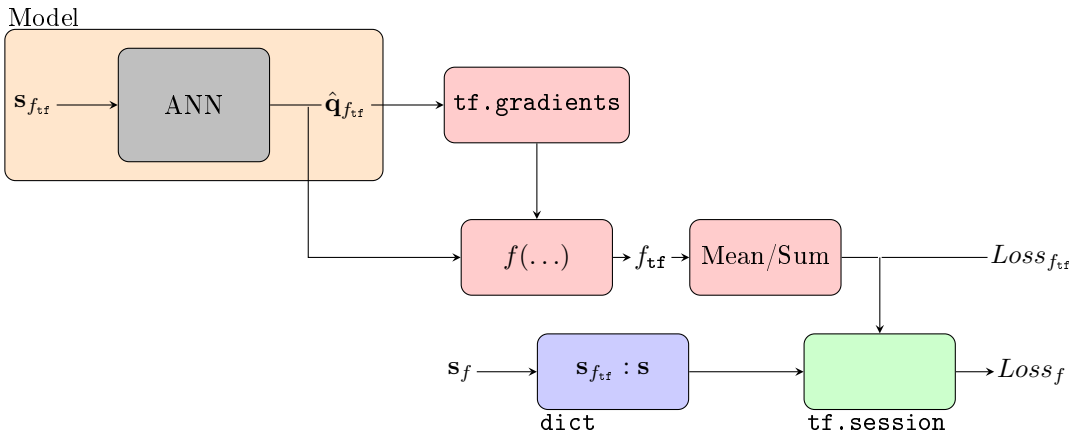


Figure 3.9: Construction of the physics loss function in `TensorFlow`

on the boundary for which the Neumann boundary condition is valid. Boundary conditions for some or all of the outputs $\mathbf{q}^{\hat{*}}$ can be employed.

3.3.7 Optimization/training strategy

The Adam optimizer is first used, with a learning rate of approximately 1×10^{-8}. After the optimizer reaches a 1000 iterations, the training switches to the L-BFGS optimizer. The Adam optimizer is built in to `TensorFlow`: `tf.train.AdamOptimizer`. Whereas the L-BFGS optimizer is taken from the `SciPy` package, by means of an external optimizer interface: `tf.contrib.opt.ScipyOptimizerInterface`. For the L-BFGS optimizer, the default parameters are used, where the convergence criterion was based on the resolution of the floating-point type used.

3.3.8 Post-processing/Predict

Once the artificial neural network is trained. The domain can be densely sampled to test the accuracy of the solution(s). It is assumed that there is a significant number of test samples such that the artificial neural network is not tested on the training data solely, as that could give a more positive performance figure than is the actual case. Moreover, next to the outputs, the gradients of all the outputs are computed over the domain, such that if knowledge is available on the data an extra means for assessing the accuracy is available. By using the governing equations, the discrepancy of the physics equations can be computed as well. This discrepancy is a combination of the errors made in the outputs and the gradients but can be insightful in assessing whether the solution given is physically consistent or not.

3.4 Workflow/Summary

In the previous sections, the framework, the scaling of the variables and equations as well as the implementation is discussed in detail. To summarize, the workflow for setting up and running the framework is presented below:

1. **Load/generate data set:** $\mathbf{S}_d \in \mathbb{R}^{N_d \times N_{in}}$ with corresponding set $\mathbf{Q}_d \in \mathbb{R}^{N_d \times N_{out}}$. The way data is generated can be by sampling uniformly or randomly (by means of LHS sampling), or by taking data from an external source.

2. **Boundary Conditions:** If extra boundary conditions are included, the data points on the boundaries are generated in this step as well as the solutions for them.

3. **Architecture:** set up artificial neural network with N_i number of inputs, N_o number of outputs, N_n number of neurons in La hidden layers. For all artificial neurons, the activation functions are chosen as the same. The weights and biases of the neural network are initialized randomly.

4. **Physics:** Subsequently, the form for the physics equations is chosen in either unit-range form (designated as NN), non-dimensional form (ND) or dimensional form (D). The N_f number of physics points is generated and sampled from the domain uniformly or randomly.

5. **Loss function:** For both the data and physics inclusion, either MSE or SSE loss functions are specified. The same applies to boundary conditions if there are any applied.

6. **Training:** The training is carried out by two optimizers, the Adam optimizer is run for 1000 iterations after which the L-BFGS optimizer is run until the optimizer does not minimize the loss function further. This is repeated for approximately 10 epochs.

7. **Predicting:** The solutions, gradients as well as the governing equations are evaluated on a uniform grid within the domain. The domain is sampled densely to about $N = 200^2$ points unless stated otherwise.

8. **Post-processing:** Plotting the solutions as well as accuracy computations are carried out in this step.

For the framework, this means that several choices are available. The optimization strategy is fixed as explained before. Also, the activation functions of the artificial neurons are fixed on the tanh function. However, the architecture (hidden layers La and number of artificial neurons per layer N_n) is to be chosen, as well as choices with regard to sampling and loss functions. The loss functions need to be chosen individually for data, physics and if used, boundary conditions. For an overview, see Table 3.1. Here, the boundary conditions are omitted as they will be excluded. It is assumed that the data will provide enough information for the problems considered, such that additional boundary conditions are not required.

Architecture	N_n, La
Data	N_d, uniform/LHS sampling
Physics	N_f, uniform/LHS sampling
Loss	MSE/SSE for Data, Physics (and Boundary conditions)

Table 3.1: Parameters to be chosen for DPINN framework

4

Fluid Flow

The governing equations for fluid flow are the Navier-Stokes equations (Section 4.1). By taking data from fluid flows (experiments/ analytical sources etc.), the data-physics fluid informed neural networks (DPFINN) can be utilized for flow variables inference. This chapter aims to give the exact forms of the fluid flow equations as well as test cases for the DPFINN framework (Section 4.4). This framework is the same as the DPINN framework, however, this framework is explicitly for fluid flow because of a modification. As a result of the incompressible flow assumption made for this framework, the divergence of the velocity field is divergence-free. The modification to the DPFINN framework is to ensure the resulting velocity fields from the artificial neural network are divergence-free (Section 4.2). The exact implementation is explained in Section 4.3.

4.1 Navier-Stokes equations

Typically, the governing quantities of fluid flows are: velocity, pressure and density as well as temperature, denoted by $\mathbf{u}$, p, ρ and T. Mathematically, they are vectors or scalar quantities that are functions of space and time ([3], [39]):

$$\mathbf{u} = \mathbf{u}(t, \mathbf{x}) \qquad p = p(t, \mathbf{x})$$
$$\rho = \rho(t, \mathbf{x}) \qquad T = T(t, \mathbf{x})$$

Where again $\mathbf{x}$ is a vector that contains the space variables and t denotes the time variable. These quantities are related through the Navier-Stokes equations, which are in gradient-form:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{u}) = 0 \tag{4.1}$$

$$\frac{\partial \rho \mathbf{u}}{\partial t} + \nabla \cdot (\rho \mathbf{u}\mathbf{u}) + \nabla p - \mu \nabla \cdot \left((\nabla \mathbf{u})^T + \nabla \mathbf{u} - \frac{1}{3} \nabla \cdot \mathbf{u} \right) = 0 \tag{4.2}$$

Where the first equation governs mass conservation and is often referred to as the continuity equation, while the second equation is the conservation of momentum. For compressible flows, ρ is an extra primary flow variable and therefore the system requires additional equations (i.e. the energy equation and equation of state) to close the system. These equations are omitted here. Fluid flow is said to be incompressible whenever the flow Mach number $Ma = \frac{U_0}{a_0}$, is large. Where U_0 is the reference or free-stream velocity and a_0 is the speed of sound. This is a consequence of the density ρ being nearly constant within the flow. With ρ = constant, $\mathbf{u}$ and p are now the primary flow variables [3], which renders the additional obsolete, as there are less variables to be solved for. The result of ρ = constant means that the time derivative, as well as the divergence of ρ, is zero, which then yields the following continuity equation:

$$\nabla \cdot \mathbf{u} = 0 \tag{4.3}$$

Which means that for mass conservation to be satisfied in an incompressible flow, the divergence of the velocity field has to be exactly zero i.e. the velocity field is a solenoidal velocity field. Because of this result, along with ρ = constant, the momentum equations can therefore be simplified with the following simplifications per term:

$$\begin{aligned}
\frac{\partial \rho \mathbf{u}}{\partial t} &= \mathbf{u}\frac{\partial \rho}{\partial t} + \rho \frac{\partial \mathbf{u}}{\partial t} \\
&= \rho \frac{\partial \mathbf{u}}{\partial t} \\
\nabla \cdot (\rho \mathbf{u}\mathbf{u}) &= \rho \nabla \cdot (\mathbf{u}\mathbf{u}) \\
&= (\nabla \cdot \mathbf{u})\,\mathbf{u} + (\mathbf{u} \cdot \nabla)\,\mathbf{u} \\
&= (\mathbf{u} \cdot \nabla)\,\mathbf{u} \\
\nabla \cdot \left((\nabla \mathbf{u})^T + \nabla \mathbf{u} - \frac{1}{3} \nabla \cdot \mathbf{u} \right) &= \nabla \cdot \nabla \mathbf{u} = \nabla^2 \mathbf{u}
\end{aligned}$$

Which yields the following incompressible Navier-Stokes equations:

$$\nabla \cdot \mathbf{u} = 0 \tag{4.4}$$

$$\rho \frac{\partial \mathbf{u}}{\partial t} + \rho\,(\mathbf{u} \cdot \nabla)\,\mathbf{u} + \nabla p - \mu \nabla^2 \mathbf{u} = 0 \tag{4.5}$$

Here, the first equation is again the divergence constraint on the velocity field, to satisfy incompressible mass conservation. The second equation is the incompressible momentum equation. The first term in

this equation is the unsteady velocity term, whereas the second term is also referred to as the convective term because it describes the transportation of the velocity gradients. The third term is the pressure gradient which can be understood as a driving term, just as a pressure difference over a pipe can push the flow through the pipe. The last term is the viscous diffusion term which describes the viscous smoothing of momentum, acting as a sink. The strength of the diffusion term is dependent on dynamic viscosity $\mu = \rho\nu$ of the fluid. Here, the reference viscosity $\mu = \mu_0 = \rho_0\nu_0$ is introduced, since it is assumed that the fluid is Newtonian and therefore the viscosity is constant.

4.1.1 Non-dimensionalization

The equations are in dimensional form, meaning that the units of space, velocity, pressure and time are all in their respective dimensions. By using reference variables L_0, t_0, U_0 and by taking $\rho = \text{constant} = \rho_0$, the flow variables can be non-dimensionalized in the following way:

$$t^* = \frac{t}{t_0} = \frac{t}{\frac{L0}{U_0}} \qquad \mathbf{x}^* = \frac{\mathbf{x}}{L_0} \qquad \mathbf{u}^* = \frac{\mathbf{u}}{U_0} \qquad p^* = \frac{p}{\rho_0 U_0^2}$$

Here, the length scale L_0 takes the role of δ_s and U_0 is one of the scales within $\mathbf{q}_0$ as described in Section 3.1. By using the chain-rule, the terms can be rewritten to their non-dimensional forms, which will then yield the non-dimensional form of the incompressible Navier-Stokes equations:

$$\nabla^* \cdot \mathbf{u}^* = 0 \tag{4.6}$$

$$\frac{\rho_0 U_0^2}{L_0}\frac{\partial \mathbf{u}^*}{\partial t^*} + \frac{\rho_0 U_0^2}{L_0}\left(\mathbf{u}^* \cdot \nabla^*\right)\mathbf{u}^* + \frac{\rho_0 U_0^2}{L_0}\nabla^* p^* - \frac{\mu_0 U_0}{L_0^2}\nabla^{*2}\mathbf{u}^* = 0 \tag{4.7}$$

Dividing by $\frac{\rho_0 U_0^2}{L_0}$ and defining the Reynolds number $Re = \frac{U_0 L_0 \rho 0}{\mu_0}$ gives:

$$\nabla^* \cdot \mathbf{u}^* = 0 \tag{4.8}$$

$$\frac{\partial \mathbf{u}^*}{\partial t^*} + \left(\mathbf{u}^* \cdot \nabla^*\right)\mathbf{u}^* + \nabla^* p^* - \frac{1}{Re}\nabla^{*2}\mathbf{u}^* = 0 \tag{4.9}$$

By non-dimensionalizing the equations, the space variables as well as Ω is transformed into Ω^*

4.1.2 Simplifications

A first simplification is that the velocity field can be only a function of space or varies relatively slow with time i.e. $\mathbf{u} = \mathbf{u}(\mathbf{x})$. This means the time-derivative of the velocity field negligible and therefore the steady incompressible Navier-Stokes equations are obtained:

$$\nabla^* \cdot \mathbf{u}^* = 0 \tag{4.10}$$

$$\left(\mathbf{u}^* \cdot \nabla^*\right)\mathbf{u}^* + \nabla^* p^* - \frac{1}{Re}\nabla^{*2}\mathbf{u}^* = 0 \tag{4.11}$$

The Navier-Stokes equations given before are valid for viscous fluids, where the velocity field is diffused through the diffusion term scaled by the dynamic viscosity μ_0. However, typically incompressible flows are characterized by the Reynolds-number Re given before. When the Reynolds number is large, the flow is typically dominated by convection. This can also be deduced from the non-dimensional Navier-Stokes equations (eq 4.8). A large Re will decrease the relative magnitude of the diffusion term. Neglecting that term (along with the unsteady term) will yield the (non-dimensional) inviscid steady Euler (momentum) equations:

$$\left(\mathbf{u}^* \cdot \nabla^*\right)\mathbf{u}^* + \nabla^* p^* = 0 \tag{4.12}$$

Where not neglecting the unsteady term, will give the unsteady form of the Euler equations, just like as for the Navier-Stokes equations. It is worth noting that this is the incompressible form. A large Reynolds number can also be present for large Mach number flows, where the flow is not to be considered incompressible anymore and the above Euler equations are invalid. Nevertheless, throughout this work, only steady and compressible flows are considered.

4.1.3 Transformation

Similarly as described in Section 3.2, a scaling factor S_c^* can be devised to transform the velocities and pressure to range close to the unit range form. In this case, information on the velocities u and v is known, in the sets $\mathbf{D}_u \in \mathbb{R}^{N_d \times 1}$ and $\mathbf{D}_v \in \mathbb{R}^{N_d \times 1}$m which are in $\mathbf{D}$. By using the reference value U_0, the sets of data can be non-dimensionalized giving $\mathbf{D}_{u^*}$ and $\mathbf{D}_{v^*}$. From these data sets, $||\mathbf{D}_{\mathbf{u}^*}|| \in \mathbb{R}^{N_d \times 1}$ can be computed, which is essentially the velocity magnitude in each datapoint. By using this, a scaling factor S_c^* can be taken as:

$$||\mathbf{D}_{\mathbf{u}^*}| = \sqrt{{\mathbf{D}_{u^*}}^2 + {\mathbf{D}_{v^*}}^2} \tag{4.13}$$

$$S_c^* = ||\mathbf{D}_{\mathbf{u}^*}||_{max} - ||\mathbf{D}_{\mathbf{u}^*}||_{min} \tag{4.14}$$

Therefore, any non-dimensional velocity can be scaled to a velocity that is in a range closer to the unit range by the scaling factor. For the pressure, the dimensional pressure p is non-dimensionalized by $p_0 = \rho U_0^2$ giving p^*. Being consistent with the reference pressure, the scaling of the non-dimensional pressure to a unit-range form is carried out by ${S_c^*}^2$. The other reason is that the transformation of the convective term results in a multiplicative term for the convective term that contains ${S_c^*}^2$, therefore by scaling the pressure by ${S_c^*}^2$, this results in only an additional multiplicative term for the viscous term. Lastly, for the non-dimensional length scale δ_s^* , a non-dimensional length scale L_s^* is taken. This is devised from the bounds of the non-dimensional domain. This domain is assumed to be rectangular with different side lengths. The average or maximum length of the sides are then used to generate L_s^*, which gives a scalar quantity.

$$L_s^* = avg(\mathbf{x}_{max}^* - \mathbf{x}_{min}^*) \qquad L_s^* = \max(\mathbf{x}_{max}^* - \mathbf{x}_{min}^*) \tag{4.15}$$

By using L_s^* and S_c^*, the scaled quantities for velocity ($\mathbf{u}^{\hat{*}}$), pressure ($p^{\hat{*}}$) and the length scale can be obtained:

$$\mathbf{x}^{\hat{*}} = \frac{\mathbf{x}^*}{L_s^*} \qquad \mathbf{u}^{\hat{*}} = \frac{\mathbf{u}^*}{S_c^*} \qquad p^{\hat{*}} = \frac{p^*}{{S_c^*}^2} \tag{4.16}$$

the equations can be scaled again, in a similar fashion as is done to obtain the non-dimensional equations. Taking equation 4.10 and applying the transformation (as well as on the gradients by using the chain-rule), the following equation is obtained

$$\frac{S_c^*}{L_s^*} \nabla^{\hat{*}} \cdot \mathbf{u}^{\hat{*}} = 0 \tag{4.17}$$

$$\frac{{S_c^*}^2}{L_s^*} \left(\mathbf{u}^{\hat{*}} \cdot \nabla^{\hat{*}} \right) \mathbf{u}^{\hat{*}} + \frac{{S_c^*}^2}{L_s^*} \nabla^{\hat{*}} p^{\hat{*}} - \frac{1}{Re} \frac{S_c^*}{{L_s^*}^2} {\nabla^{\hat{*}}}^2 \mathbf{u}^{\hat{*}} = 0 \tag{4.18}$$

Dividing by $\frac{{S_c^*}^2}{L_s^*}$ and noting that both the scales are non-zero $L_s^* \neq 0$ and $S_c^* \neq 0$ gives the following scaled equations

$$\nabla^{\hat{*}} \cdot \mathbf{u}^{\hat{*}} = 0 \tag{4.19}$$

$$\left(\mathbf{u}^{\hat{*}} \cdot \nabla^{\hat{*}} \right) \mathbf{u}^{\hat{*}} + \nabla^{\hat{*}} p^{\hat{*}} - \frac{1}{Re} \frac{1}{L_s^* S_c^*} {\nabla^{\hat{*}}}^2 \mathbf{u}^{\hat{*}} = 0 \tag{4.20}$$

A result of the scaling is that the viscous term is scaled by both the scales, nevertheless the equation is consistent with the original dimensional equations. This yields the third and last set of equations, the dimensional equations are referred to as the *D* (dimensional) equations, the non-dimensional are denoted by *ND* and these scaled ones are *NN*. While the length scale does not strictly bring $\mathbf{x}^*$ in the $[-1, 1]$ range, it at least ensures that it is within $[0, 1]$.

4.1.4 Coordinate system

While not strictly a requirement for the framework. In the following, Cartesian coordinate systems are only considered, therefore the space variable $\mathbf{x}$ is $\mathbf{x} = (x, y)^T$ and $\mathbf{x} = (x, y, z)^T$ for $\mathbb{R}^2$ and $\mathbb{R}^3$ spaces respectively. Furthermore, a confined space $\Omega \subset \mathbb{R}^n$ is considered, which is bounded by $\partial\Omega$. It is assumed that Ω does not change as a function of time. Since Cartesian coordinate systems are used, the components of the velocity vector $\mathbf{u}$ are $\mathbf{u} = (u, v)$ for 2D problems ($n = 2$) and $\mathbf{u} = (u, v, w)$ for 3D problems ($n = 3$). The expanded forms of the $n = 2$ Navier-Stokes and Euler equations for the Cartesian coordinate system as well as for all the three physics forms (*D*-form, *ND*-form and *NN*-form) are shown below.

Steady (incompressible) Navier-Stokes equations

D-form

$$\begin{aligned} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 \\ u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial x} - \nu\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) &= 0 \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial y} - \nu\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) &= 0 \end{aligned} \tag{4.21}$$

ND-form

$$\begin{aligned} \frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} &= 0 \\ u^*\frac{\partial u^*}{\partial x^*} + v^*\frac{\partial u^*}{\partial y^*} + \frac{\partial p^*}{\partial x^*} - \frac{1}{Re}\left(\frac{\partial^2 u^*}{\partial x^{*2}} + \frac{\partial^2 u^*}{\partial y^{*2}}\right) &= 0 \\ u^*\frac{\partial v^*}{\partial x^*} + v^*\frac{\partial v^*}{\partial y^*} + \frac{\partial p^*}{\partial y^*} - \frac{1}{Re}\left(\frac{\partial^2 v^*}{\partial x^{*2}} + \frac{\partial^2 v^*}{\partial y^{*2}}\right) &= 0 \end{aligned} \tag{4.22}$$

NN-form

$$\begin{aligned} \frac{\partial u^{\hat{*}}}{\partial x^{\hat{*}}} + \frac{\partial v^{\hat{*}}}{\partial y^{\hat{*}}} &= 0 \\ u^{\hat{*}}\frac{\partial u^{\hat{*}}}{\partial x^{\hat{*}}} + v^{\hat{*}}\frac{\partial u^{\hat{*}}}{\partial y^{\hat{*}}} + \frac{\partial p^{\hat{*}}}{\partial x^{\hat{*}}} - \frac{1}{Re}\frac{1}{L_s^* S_c^*}\left(\frac{\partial^2 u^{\hat{*}}}{\partial x^{\hat{*}2}} + \frac{\partial^2 u^{\hat{*}}}{\partial y^{\hat{*}2}}\right) &= 0 \\ u^{\hat{*}}\frac{\partial v^{\hat{*}}}{\partial x^{\hat{*}}} + v^{\hat{*}}\frac{\partial v^{\hat{*}}}{\partial y^{\hat{*}}} + \frac{\partial p^{\hat{*}}}{\partial y^{\hat{*}}} - \frac{1}{Re}\frac{1}{L_s^* S_c^*}\left(\frac{\partial^2 v^{\hat{*}}}{\partial x^{\hat{*}2}} + \frac{\partial^2 v^{\hat{*}}}{\partial y^{\hat{*}2}}\right) &= 0 \end{aligned} \tag{4.23}$$

Steady (incompressible) Euler equations

D-form

$$\begin{aligned} \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} &= 0 \\ u\frac{\partial u}{\partial x} v\frac{\partial u}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial x} &= 0 \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{1}{\rho}\frac{\partial p}{\partial y} &= 0 \end{aligned} \tag{4.24}$$

ND-form

$$\begin{aligned} \frac{\partial u^*}{\partial x^*} + \frac{\partial v^*}{\partial y^*} &= 0 \\ u^*\frac{\partial u^*}{\partial x^*} + v^*\frac{\partial u^*}{\partial y^*} + \frac{\partial p^*}{\partial x^*} &= 0 \\ u^*\frac{\partial v^*}{\partial x^*} + v^*\frac{\partial v^*}{\partial y^*} + \frac{\partial p^*}{\partial y^*} &= 0 \end{aligned} \tag{4.25}$$

NN-form

$$\begin{aligned} \frac{\partial u^{\hat{*}}}{\partial x^{\hat{*}}} + \frac{\partial v^{\hat{*}}}{\partial y^{\hat{*}}} &= 0 \\ u^{\hat{*}}\frac{\partial u^{\hat{*}}}{\partial x^{\hat{*}}} + v^{\hat{*}}\frac{\partial u^{\hat{*}}}{\partial y^{\hat{*}}} + \frac{\partial p^{\hat{*}}}{\partial x^{\hat{*}}} &= 0 \\ u^{\hat{*}}\frac{\partial v^{\hat{*}}}{\partial x^{\hat{*}}} + v^{\hat{*}}\frac{\partial v^{\hat{*}}}{\partial y^{\hat{*}}} + \frac{\partial p^{\hat{*}}}{\partial y^{\hat{*}}} &= 0 \end{aligned} \tag{4.26}$$

4.2 Solenoidal Velocity Field - DPFINN

A result of the incompressible flow assumption is that the divergence of the flow velocity field is always zero: $\nabla \cdot u = 0$ where u is a vector field, A divergence-free velocity field is also referred to as being a solenoidal velocity field. One way to reconstruct a solenoidal velocity field is by using a basis that by itself yields a divergence-free velocity field. For example, a potential $\mathbf{q}$ can be used where twice the curl of the potential will yield the velocity field $\mathbf{u}$. Then, by definition, as the divergence of the curl is always zero, the resulting velocity fields will then also be divergence-free:

$$\mathbf{u} = \nabla \times \nabla \times \mathbf{q} \tag{4.27}$$

$$\nabla \cdot \mathbf{u} = \nabla \cdot (\nabla \times \nabla \times \mathbf{q}) = 0 \tag{4.28}$$

This essentially follows the approach suggested by [4] and [32]. Here, the potential vector has the same dimension as the velocity vector: if $\mathbf{u} \in \mathbb{R}^n$ then $\mathbf{q} \in \mathbb{R}^n$ as well. Following [49], the equation can also be rewritten as:

$$\mathbf{u} = \left(-\Delta + \nabla\nabla^T\right)\mathbf{q} \tag{4.29}$$

If a cartesian coordinate system is used and $n = 2$, then the operator on $\mathbf{q} \in \mathbb{R}^n$ can be rewritten as:

$$(\Delta I - \nabla\nabla^T) = \left(\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\right)\begin{pmatrix}1 & 0\\ 0 & 1\end{pmatrix} - \begin{pmatrix}\frac{\partial}{\partial x}\\ \frac{\partial}{\partial y}\end{pmatrix}\begin{pmatrix}\frac{\partial}{\partial x} & \frac{\partial}{\partial y}\end{pmatrix} \tag{4.30}$$

$$= \begin{pmatrix}\frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} & 0\\ 0 & \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2}\end{pmatrix} - \begin{pmatrix}\frac{\partial^2}{\partial x^2} & \frac{\partial^2}{\partial x \partial y}\\ \frac{\partial^2}{\partial y \partial x} & \frac{\partial^2}{\partial y^2}\end{pmatrix} \tag{4.31}$$

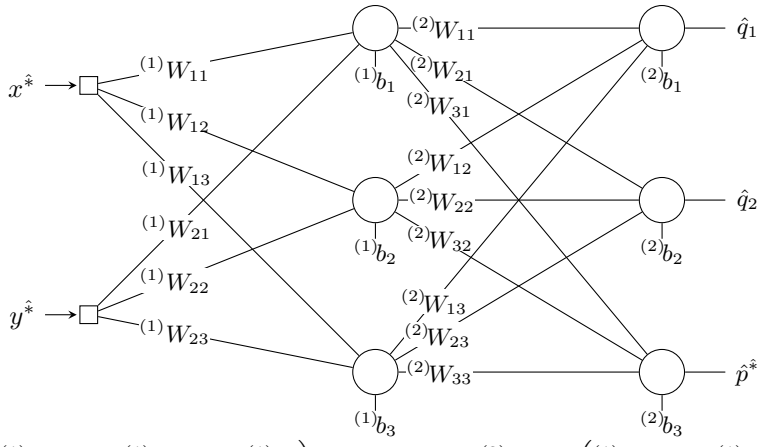
$$= \begin{pmatrix}-\frac{\partial^2}{\partial y^2} & \frac{\partial^2}{\partial x \partial y}\\ \frac{\partial^2}{\partial y \partial x} & -\frac{\partial^2}{\partial x^2}\end{pmatrix} \tag{4.32}$$

$$\tag{4.33}$$

And therefore if $\mathbf{q} = (q_1, q_2)^T$ the velocity vector $\mathbf{u} \in \mathbb{R}^2$ becomes:

$$\mathbf{u} = \begin{pmatrix}u\\ v\end{pmatrix} = \begin{pmatrix}-\frac{\partial^2 q_1}{\partial y^2} + \frac{\partial^2 q_2}{\partial x \partial y}\\ \frac{\partial^2 q_1}{\partial y \partial x} - \frac{\partial^2 q_2}{\partial x^2}\end{pmatrix} \tag{4.34}$$

This potential will be used in the framework of the artificial neural networks, which can then be differentiated to obtain the velocities in both components. So, it is not a coincidence that the potential is named $\mathbf{q}$, as the first two outputs of the artificial neural network will be reserved for the velocity potential. The third one will still be the pressure $\hat{p^*}$, thus for 2D problems: $\hat{\mathbf{q}} = (\hat{q}_1, \hat{q}_2, \hat{p^*})$. By using this basis for the velocity, the bias terms of the output nodes are obsolete. The only one that could be of importance is the one for the pressure output $\hat{p^*}$. The bias here adds or subtracts to the overall pressure field. However, since incompressible flows are considered, the pressure is defined up to a constant, so that renders that bias term also unimportant. To show this, a similar very simple artificial neural network is considered: ($N_n = 3$, $N_i = 2$, $La = 1$, $N_o = 3$). Clearly, differentiating either $\hat{q}_1$ or $\hat{q}_2$ will immediately drop the constant biases ${}^{(2)}b_1$ and ${}^{(2)}b_2$ respectively, regardless of what it is differentiated to. Furthermore, the constant bias ${}^{(2)}b_3$ provides the constant shifting in the pressure field and can be adjusted a posteriori to have the pressure field match to a certain reference pressure at a certain point. However, this is not different from just adding or subtracting a reference pressure.



$$\hat{q}_1 = {}^{(2)}W_{11}\sigma\left({}^{(1)}W_{11}x + {}^{(1)}W_{12}y + {}^{(1)}b_1\right) + {}^{(2)}W_{12}\sigma\left({}^{(1)}W_{21}x + {}^{(1)}W_{22}y + {}^{(1)}b_2\right) + {}^{(2)}W_{13}\sigma\left({}^{(1)}W_{31}x + {}^{(1)}W_{32}y + {}^{(1)}b_3\right) + {}^{(2)}b_1$$

$$\hat{q}_2 = {}^{(2)}W_{21}\sigma\left({}^{(1)}W_{11}x + {}^{(1)}W_{12}y + {}^{(1)}b_1\right) + {}^{(2)}W_{22}\sigma\left({}^{(1)}W_{21}x + {}^{(1)}W_{22}y + {}^{(1)}b_2\right) + {}^{(2)}W_{23}\sigma\left({}^{(1)}W_{31}x + {}^{(1)}W_{32}y + {}^{(1)}b_3\right) + {}^{(2)}b_2$$

Another result of using the divergence-free basis is that to obtain the velocity field $\hat{\mathbf{u}}^{\hat{*}}$, the potentials $\hat{\mathbf{q}}$ need to be differentiated twice. Therefore, not only the first derivative but also the second derivative is required of the activation function. Moreover, if the momentum equations are evaluated, even higher-order derivatives of the activation function are required. To illustrate this, the expression for $\hat{\mathbf{u}}^{\hat{*}}$ (equation 4.34) is substituted into the u-momentum equation:

$$\begin{aligned}
u^{\hat{*}}\frac{\partial u^{\hat{*}}}{\partial x^{\hat{*}}} + v^{\hat{*}}\frac{\partial u^{\hat{*}}}{\partial y^{\hat{*}}} + \frac{\partial p^{\hat{*}}}{\partial x^{\hat{*}}} &= 0 \\
\left(-\frac{\partial^2 q_1}{\partial y^{\hat{*}2}} + \frac{\partial^2 q_2}{\partial x^{\hat{*}}\partial y^{\hat{*}}}\right)\frac{\partial}{\partial x^{\hat{*}}}\left(-\frac{\partial^2 q_1}{\partial y^{\hat{*}2}} + \frac{\partial^2 q_2}{\partial x^{\hat{*}}\partial y^{\hat{*}}}\right) + \left(\frac{\partial^2 q_1}{\partial y^{\hat{*}}\partial x^{\hat{*}}} - \frac{\partial^2 q_2}{\partial x^{\hat{*}2}}\right)\frac{\partial}{\partial y^{\hat{*}}}\left(-\frac{\partial^2 q_1}{\partial y^{\hat{*}2}} + \frac{\partial^2 q_2}{\partial x^{\hat{*}}\partial y^{\hat{*}}}\right) + \frac{\partial p^{\hat{*}}}{\partial x^{\hat{*}}} &= 0 \\
\left(-\frac{\partial^2 q_1}{\partial y^{\hat{*}2}} + \frac{\partial^2 q_2}{\partial x^{\hat{*}}\partial y^{\hat{*}}}\right)\left(-\frac{\partial^3 q_1}{\partial x^{\hat{*}}\partial y^{\hat{*}2}} + \frac{\partial^3 q_2}{\partial x^{\hat{*}2}\partial y^{\hat{*}}}\right) + \left(\frac{\partial^2 q_1}{\partial y^{\hat{*}}\partial x^{\hat{*}}} - \frac{\partial^2 q_2}{\partial x^{\hat{*}2}}\right)\left(-\frac{\partial^3 q_1}{\partial y^{\hat{*}3}} + \frac{\partial^3 q_2}{\partial x^{\hat{*}}\partial y^{\hat{*}2}}\right) + \frac{\partial p^{\hat{*}}}{\partial x^{\hat{*}}} &= 0
\end{aligned} \tag{4.35}$$

A similar result can be obtained for the v-momentum equations. Which shows that the momentum equations are a mixture of (mixed) higher-order derivatives. It is worth noting that, since training uses gradient-based optimization, the equations are differentiated again for the physics loss function.

4.3 Data-Physics Fluids Informed Neural Networks

The DPINN applied to the fluid flow test cases results in the data-physics fluids informed neural network (DPFINN). Here, the artificial neural network takes as inputs the coordinates in unit range form and outputs the vector for the solenoidal basis as well as the pressure (Figure 4.1). By subsequently evaluating the gradients of $\hat{\mathbf{u}}^{\hat{*}}_{\mathtt{tf}}$ and $\hat{p}^{\hat{*}}_{\mathtt{tf}}$, the divergence and the momentum equations can be obtained. These will be used for the physics loss function while the data loss function can also be obtained (see subsection 3.3.6).

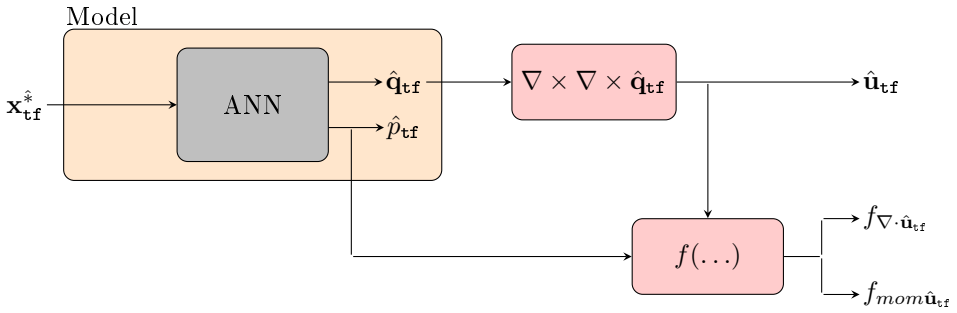


Figure 4.1: Schematic of DPFINN framework in `TensorFlow`

For a DPFINN application, there are a significant number of parameters and choices to choose from. All of them are linked and have their effect on the outcomes. First of all, the number of inputs is fixed to the number of coordinates in the space, in this case, there are two: $x^{\hat{*}}$ and $y^{\hat{*}}$ so therefore, $N_i = 2$. Furthermore, two outputs are reserved for the solenoidal basis and one for the pressure: $N_o = 3$. Moreover, the activation function is fixed to the $\sigma = \tanh$ activation function. The reason for this is, first of all, to reduce the number of parameters and secondly because the activation function is smooth and also used in [36]. This leaves the number of artificial neurons N_n and the number of layers La as choices for the architecture specification (Figure 4.2).

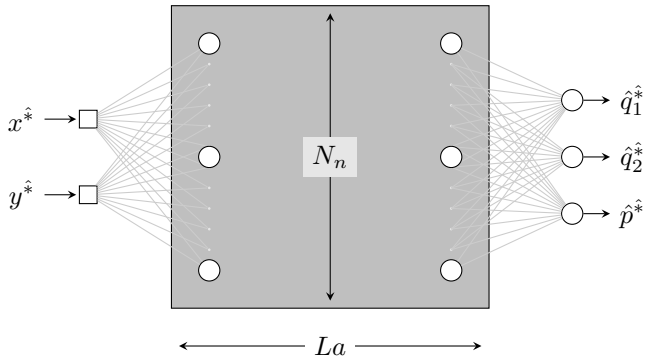


Figure 4.2: General artificial neural network parameters (N_i, N_n, N_o, La) schematic. The hidden layers are drawn in the gray box.

Concerning the data sampling, this depends on the case and therefore N_d remains a parameter as well as the sampling strategy (uniform or randomly). For the physics collocation points, they are generated randomly by the LHS method and typically $N_f = 10^2$ or $N_f = 100^2$. To these, the datapoints are added, therefore the total number of physics collocation points are $N_f = 10^2 + N_d$ or $N_f = 100^2 + N_d$. But throughout the remainder, N_f will be referred to as the additional physics collocation points.

4.4 Cases

For the testing of the DPFINN framework, a collection of four test cases are devised. All the test cases are 2D steady flow cases and for three of them, analytical solutions are available. For two of the test cases, the flow is inviscid, so the Euler equations are applicable. These flow cases are a inviscid stagnation flow (4.4.1) and a single Taylor-Green vortex (4.4.3). Furthermore, two viscous cases are considered: the viscous stagnation flow (also known as Hiemenz stagnation flow, see subsection 4.4.2) and the lid-driven cavity flow (4.4.4). The next three subsections describe the analytical solutions for the first three cases, while for the latter, the OpenFOAM solution is used and described in the subsection after. Moreover, it is noteworthy that all test cases are incompressible, thus satisfy the incompressible equations presented in Section 4.1. This also means that all the test cases have solenoidal (divergence-free) velocity fields, as they satisfy equation 4.3.

4.4.1 Inviscid Stagnation Flow

The flow in this test case is an incompressible and inviscid flow impinging 2D steady flow on a wall. The wall is located horizontally at $y = 0$ and the domain is taken as $\Omega = [0, L_0] \times [0, L_0]$ (see Figure 4.6. Here L_0 is the reference length scale and therefore is δ_s in this case. The origin is the stagnation point where $u = v = 0$. Since the flow is incompressible, the continuity equation can be satisfied by a plane stream function:

$$u = \frac{\partial \psi}{\partial y} \qquad v = -\frac{\partial \psi}{\partial x} \tag{4.36}$$

The solution then is $\psi = Bxy$, where B is a constant. Then immediately, the velocity can be found from the streamfunction by integrating the streamfunctions. Moreover, the Bernoulli equation can be utilized to determine the pressure (equations 4.37).

Inviscid Stagnation Flow

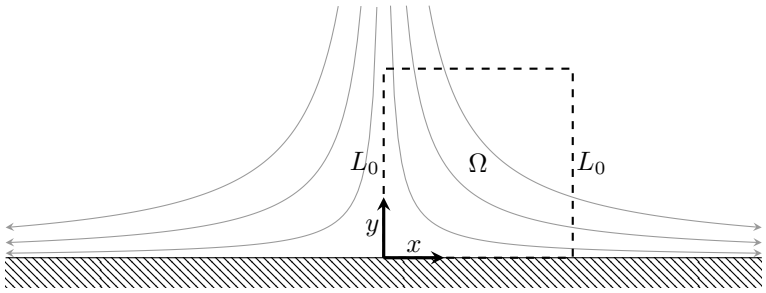


Figure 4.3: Stagnation flow domain

$$\begin{gathered} u(x,y) = Bx \qquad v(x,y) = -By \\ p(x,y) = p_0 - \frac{1}{2}\rho_0(u^2+v^2) = p(0,0) - \frac{1}{2}\rho_0 B^2(x^2+y^2) \end{gathered} \tag{4.37}$$

$$L_0 = 1.3 \qquad U_0 = 4.05 \qquad \rho_0 = 1.0 \qquad B = 0.8\frac{U_0}{L_0} \tag{4.38}$$

From the equations, it is clear that the u and v velocities are zero at the origin and therefore, the stagnation point is obtained. This is also where the stagnation pressure p_0 is attained, which is the highest pressure in the domain. Further away from the origin, the $u-$velocity increases but slips at the boundary. The $v-$velocity increases towards the wall, such that it is zero at the wall and no flow goes through the wall. The pressure decreases outwards, as a result of increased velocity magnitude. The inviscid stagnation flow satisfies the steady Euler equations (which could also be an alternative way to determine the pressure solution) [50]. The free-stream parameters are chosen as in equations 4.41. They are chosen such that when non-dimensionalized, the velocity and pressure are not necessarily zero, for testing purposes. This is achieved by scaling B by 80%. However, the rectangular domain is chosen

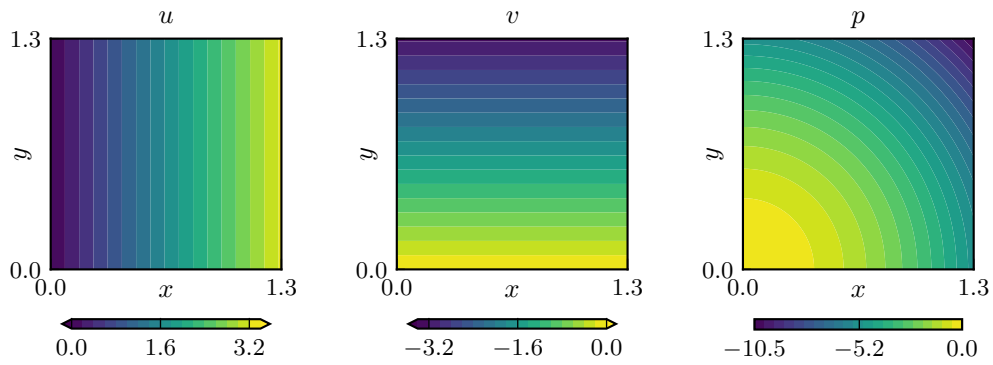


Figure 4.4: Dimensional solutions (u, v, p) of inviscid stagnation flow

according to the characteristic length L_0. Here the non-dimensional length scale will be $L_s^* = 1$. The other scaling factor S_c^* depends on the dataset of the velocity field.

4.4.2 Hiemenz Viscous Stagnation Flow

This test case is similar to the inviscid stagnation flow, however, in this case the viscosity is included. This results in no slip along the wall $u(x, 0) = 0$, (there is slip in the inviscid case). This can be modelled as is described by Hiemenz [18] and in [50], where the streamfunction is modified to include a function $f(y)$: $\psi = Bxf(y)$. By using equations 4.36, the velocities as a function of x and $f(y)$ become:

$$u(x, y) = Bxf(y) \qquad v(x, y) = -Bx\frac{df}{dy} \tag{4.39}$$

By substitution in the 2D momentum equations, the following differential equation is obtained:

$$f''' + \frac{B}{\nu}\left(ff'' - f'^2\right) = -\frac{B}{\nu}$$

As White notes in [50], there exists a length scale $\sqrt{\nu/B}$ as well as a velocity scale $\sqrt{\nu B}$. By using this length scale and a dimensionless variable $\eta = y\sqrt{\frac{B}{\nu}}$, gives the following variable-free differential equation.

$$F''' + FF'' + 1 - F'^2 = 0$$

For more details, the reader is referred to [50] and [39], where the solutions for F, F' and F'' as a function of η are given in tables, as only a numerical solution exists for the differential equation. These tables are interpolated to allow for the solution to be sampled between data points. The interpolation is carried out by a least-squares polynomial fit of degree 10. Which gives a fit as shown qualitatively in Figure 4.5. The fit is considered accurate and therefore, these interpolants can be used for F, and F', to obtain the velocity and pressure fields:

Hiemenz viscous Stagnation Flow

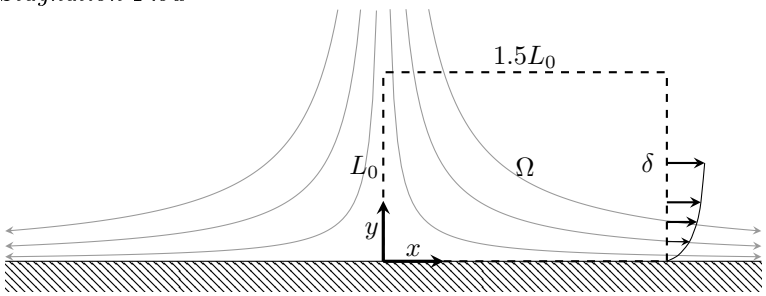


Figure 4.6: Stagnation flow domain

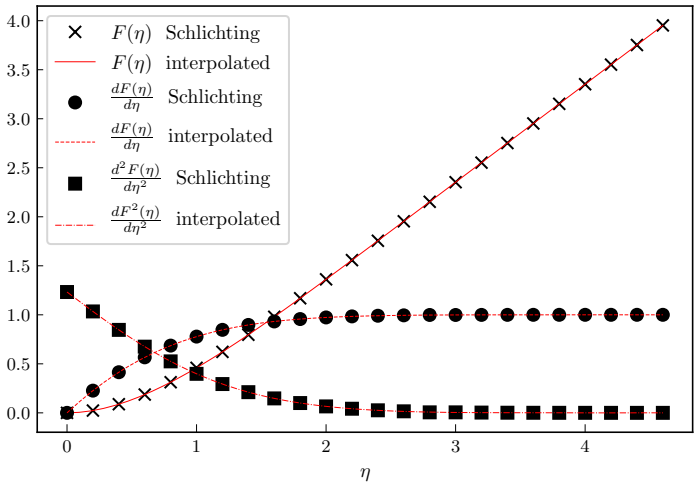


Figure 4.5: Solutions of F, F' and F'' from [39], interpolated

$$u(x,y) = BxF'(\eta) \qquad v(x,y) = -F(\eta)\sqrt{B\nu}$$
$$p(x,\eta) = p(0,0) - \frac{1}{2}\rho_0 BxF'(\eta) - \frac{1}{2}B\mu F(\eta)^2 - B\mu F'(\eta) \tag{4.40}$$

$$L_0 = 1.3 \qquad U_0 = 4.05 \qquad \rho_0 = 1.0 \qquad B = 0.8\frac{U_0}{L_0}$$
$$\nu_0 = B\left(\frac{\delta}{2.4}\right)^2 \qquad \eta_{max} = 4.6 \qquad \delta = \frac{2.4L_0}{\eta_{max}} \tag{4.41}$$

The same parameters as for the inviscid stagnation flow case are chosen. With the only change being the size of the domain. To test a non-square domain, the length of the domain in x-direction is increased to $1.5L_0$ Also, since this is a viscous case, the maximum velocity in the domain is reached only at the top right corner of the domain, as a boundary layer is present, which is approximately $u = 1.2U_0$. Similarly, the pressure is altered in similarly, as boundary like behavior seems to exist in the pressure as well [50]. The boundary layer thickness is approximately δ (not to be confused with the scaling factors δ_s and δ_s^8) is roughly half of the domain height: $\delta \approx \frac{L_0}{2}$.

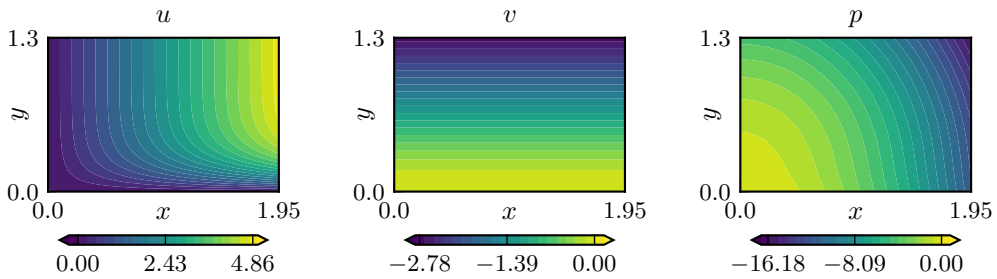


Figure 4.7: Dimensional solutions (u, v, p) of Hiemenz viscous stagnation flow

This flow satisfies the steady viscous Navier-Stokes equations, where the inviscid Euler equations are valid for the inviscid stagnation flow.

4.4.3 Taylor-Green vortex

This is a test case taken from [4]. It is a vortex that generates an incompressible velocity field in 2D. Therefore, by definition, it is divergence-free. It is a vortex that starts with an initial strength that

decays in time. Therefore, it is a problem-dependent on space ($\mathbf{x} = (x, y)$) and time t. The test case has analytical solutions which are given in equation 4.42. Here, H is the angular momentum in the vortex. The solution satisfies the unsteady Navier Stokes equations. However, the unsteady term for the velocity exactly balances the viscosity term. Therefore, for an instantaneous velocity field, the steady Euler equations can be used to infer the pressure field from the velocity field given.

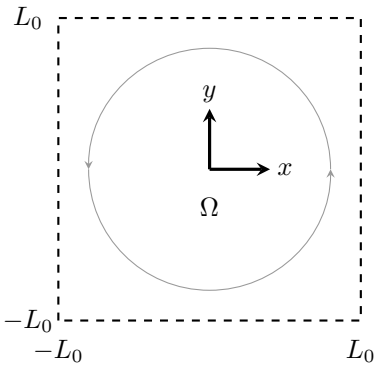


Figure 4.8: Taylor-Green vortex domain

$$
\begin{aligned}
u_\theta &= \frac{H}{8\pi}\frac{1}{\nu_0 t^2}\left(\sqrt{x^2+y^2}\right)e^{-\frac{x^2+y^2}{4\nu_0 t}} \\
u(x,y) &= -u_\theta \sin\left(\arctan\left(\frac{x}{y}\right)\right) \\
v(x,y) &= -u_\theta \cos\left(\arctan\left(\frac{x}{y}\right)\right) \\
p(x,y) &= \frac{-\rho_0 H^2}{64\pi^2\nu_0 t^3}e^{-\frac{\sqrt{x^2+y^2}^2}{2\nu_0 t}}
\end{aligned}
\tag{4.42}
$$

$$
t = 0.05 \qquad H = 1\times10^{-6} \qquad L_0 = \sqrt{H} = 1\times10^{-3} \qquad U_0 = 1\times10^{-3} \tag{4.43}
$$

$$
\rho_0 = 1000 \qquad \nu_0 = 1\times10^{-6} \qquad Re_L = \frac{U_0 L_0}{\nu_0} = 1
$$

For this test case, the same parameters are used as done in [4]. Furthermore, a velocity field at a single time instance is used ($t = 0.05$). The domain is also taken the same as $\Omega = [-L_0, L_0] \times [-L_0, L_0]$. This results in the velocity fields and the pressure field as given by Figure 4.9. Visible is that the velocity field yields a vortex in the middle of the domain rotating counter-clockwise, which induces a pressure field that has a single low value in the core of the vortex. Furthermore, the Reynolds number is very low, indicating that the vortex is a laminar flow.

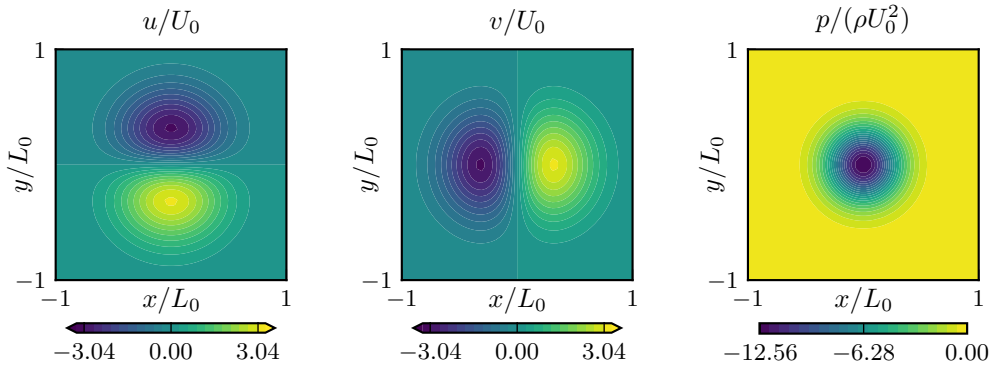


Figure 4.9: Non-dimensional solutions of the Taylor-Green vortex

4.4.4 Lid Driven Cavity

A common test case for fluid flow solvers is the lid-driven cavity flow. The domain is a 2D square box, where all walls are solid boundaries except for the top boundary. The top boundary so to say moves with a velocity from left to right. The friction of the lid causes a clockwise rotational flow in the domain. There is no analytical solution for this case, so an OpenFOAM CFD simulation is carried out. The domain is $\Omega[0,1] \times [0,1]$, where a fine grid of 200×200 cells are uniformly distributed over the domain. The flow is solved by the incompressible laminar Navier-Stokes solver using the PISO algorithm. The solutions is transient but reaches steady-state well before the final time of $t_{final} = 100$, where timesteps of $dt = 0.01$ are used. The reference velocity is the velocity of the lid ($U_0 = 1$), which is also the initial condition for the solver. The rest of the walls (south, west, and east) have no-slip boundary conditions for the velocity. For the pressure, the zero gradient (Neumann) boundary conditions are used on all boundaries. The viscosity ν was chosen depending on the Reynolds number. The Reynolds numbers considered is $Re = 100$, and since the reference velocity $U_0 = 1$ and $L_0 = 1$, the viscosity is simply: $\nu_0 = 1/Re$. The case is run until the steady-state is achieved. The velocity field after the final timestep is then extracted from the grid which can be sampled from the dataset. To keep the Courant–Friedrichs–Lewy (CFL) number below 1, the timestep is taken as $\delta t = 0.005$ as the Courant number is defined as $CFL = \frac{\delta_x |U|}{\delta_t}$ and where $\delta_x = 1/200 = 0.005$ and $|U| = 1$. Keeping $CFL \leq 1$ ensures numerical stability because the information is not allowed to travel further than one cell within one timestep. (see e.g. [2] or [30])

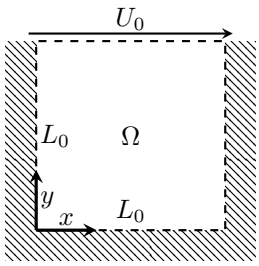


Figure 4.10: Lid-driven cavity flow domain

$$L_0 = 1 \qquad U_0 = 1 \qquad \rho_0 = 1000 \qquad \nu_0 = \frac{1}{Re} \tag{4.44}$$

This test case is meant to show that the source of the velocity fields is irrelevant in principle, as the source for the velocity field for this case is generated by OpenFOAM. Once a steady-state has been reached by the OpenFOAM solution. The results are post-processed in ParaView. Here, a $x - y$slice is made in the, from which the datapoints and the velocity data is obtained. The reason this is done is because the OpenFOAM mesh is always 3D, even though the mesh only contains one cell in the z-direction. The solutions for the velocity and pressure, for $Re = 100$ is shown in Figure 4.11, which are shown in contour plots and profiles. Where the first row in the profiles shows the locations of the profiles, the second row the u-velocity, the third row the v-velocity and the bottom row, the pressure p. To verify the results obtained from OpenFOAM, the velocity profiles are compared to the solutions given by [14]. Which are also shown in profiles. The agreement is considered to be good which indicates that indeed the steady-state solution is obtained, are comparable to literature are therefore this case is usable as a testcase. Because the domain and the velocities are already roughly in the unit ranges, it is expected that there will be little difference between the different physics implementations. The discontinuities at the top left and right corner could, however, prove to be troublesome for the framework.

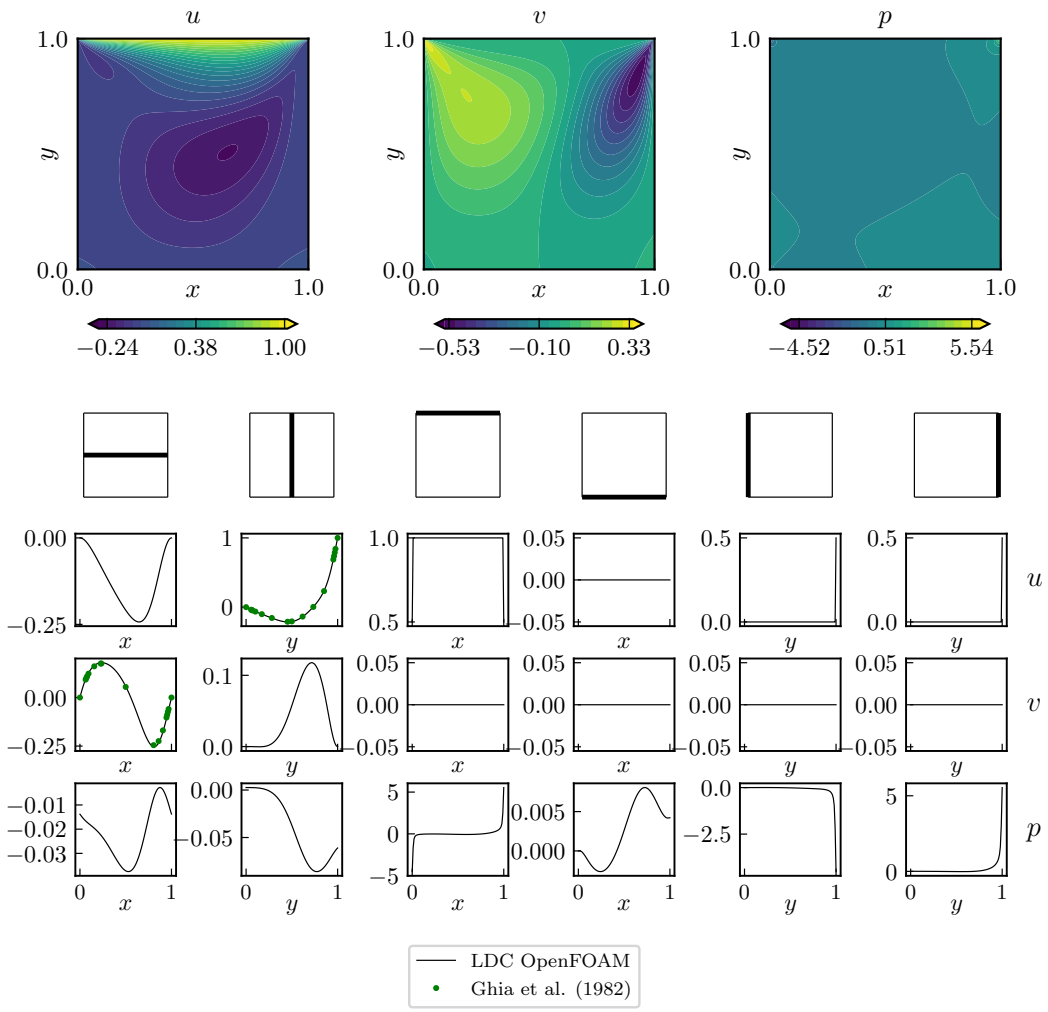


Figure 4.11: Profiles of Lid-driven cavity flow, $Re = 100$, generated by OpenFOAM and compared to [14]

5

Results

In this chapter, results for the application of the test cases from Section 4.4 to the framework described in Chapter 3 are discussed. More specifically, the test cases considered are the inviscid stagnation flow, the viscous (Hiemenz) stagnation flow, the Taylor-Green vortex, and a lid-driven cavity flow test case. First, the performance of the divergence-free basis in the DPFINN framework is tested (Section 5.1). Next, the effect of the artificial neural network complexity (Section 5.2), data and physics domain sampling (Section 5.3), physics implementations and loss function types are discussed (Section 5.4). The results are generated with varying artificial neural network architectures, varying sampling sizes, and different physics implementations as well as different loss function combinations. The goal of varying these parameters is to gain insights into the implications of the different choices. Due to a large number of parameters involved, a large parameter study could not be carried out for all of the test cases. Instead, the results are generated with specific artificial neural networks, loss function combinations, and physics collocation points.

5.1 Divergence-free basis performance

As described in 4.2, the divergence free basis is implemented by taking the potential $\hat{\mathbf{q}} = (\hat{q}_1, \hat{q}_2)^T$ and the pressure $\hat{p}$ as the artificial neural network outputs, by which the velocity field $\hat{\mathbf{u}}$ is obtained by taking twice the curl of the potential, as in equation 4.34. Regardless of the test case and regardless of any other parameter or the artificial neural network used, the divergence of the velocity field from the framework should be zero. However, due to the floating-point type precision and a relatively large amount of multiplications involved, some variations can occur. These variations should be approximately the resolution of the float point type used. For the results here the `tf.float32` and `tf.float64` types are used. Therefore, the resolutions are 10^{-7} and 10^{-16} respectively. And therefore, the maximum absolute deviations of the divergence of the velocity field should be of the same order. For the remainder of this section, each test case from Section 4.4 is used to compare the DPFINN framework results with and without a divergence-free basis.

5.1.1 Inviscid Stagnation Flow

As a first example, the inviscid stagnation flow case is taken, where the amount of data sampling is $N_d = 10^2$ (uniformly sampled), additional collocation points $N_f = 100^2$ (randomly sampled) and a sufficiently complex artificial neural network of $N_n = 10$ and $La = 3$. This case, with these settings, are run for DPFINN with and without the divergence-free basis. The settings and the artificial neural network is believed to give a good basis for evaluating the performance of the divergence-free basis, as the inviscid stagnation flow is deemed to be a reasonably simple case. The results for the case without a divergence-free basis are shown in Figure 5.1 while the results for this case with divergence-free basis is shown in Figure 5.1. The first rows show the contour plots for the velocity, pressure and velocity magnitude, while the second rows show the accuracies for these, in the dimensionless form (hence the asterisks). The bottom rows show the absolute values for the dimensional momentum equations ($f_{\hat{u}}$, $f_{\hat{v}}$) and the divergence of the framework velocity field output ($f_{\nabla\cdot\hat{\mathbf{u}}}$). Clearly, the divergence-free basis creates a velocity field of which the divergence is of the order $O(10^{-15})$, equal to the resolution of the floating-point type used: `tf.float64`. The divergence-free basis also gives velocity fields that are slightly more in agreement with the exact velocity fields. This can be seen in the plots for the velocity magnitude discrepancy. The Euler momentum equations for the results without the divergence-free show that the maximum absolute discrepancy is of the order $O(10^{-2})$, which is also the case for the momentum equation discrepancy of the run without the basis. Therefore, at least for the inviscid case, the divergence-free basis is believed to yield results that are similar to the results from the framework without the divergent-free basis. Except for the divergence of the inferred velocity field, which is near zero in contrary to the case where no divergence-free basis is used, where the maximum divergence in the domain is of the order $O(10^{-2})$.

5.1.2 Taylor-Green vortex

This case is one where the solution satisfies the Euler equations and where most of the non-zero features are centered in the middle of the domain, at the core of the vortex. This case is also incompressible, so the divergence of the velocity field should be divergence-free. The solutions from the DPFINN framework, without and with divergence-free basis employed are shown in Figures 5.3 and 5.2 respectively. Here again, $N_d = 10^2$ (uniformly sampled), $N_f = 100^2$, $N_n = 10$, $La = 3$ and the Euler equations in NN-form is used. Both are able to infer the flow variables with approximately the same accuracy, where the one divergence-free basis is believed to perform slightly better, after inspection of the discrepancy plots in the second row. For example, the divergence-free basis almost halves the non-dimensional velocity magnitude discrepancy ($|||\hat{\mathbf{u}}^*|| - ||\mathbf{u}^*|||$). Again, the maximum absolute divergence of the inferred velocity field is reduced significantly to $O(10^{-14})$, which is one order of magnitude higher than for divergence of the ISF case. The reason for this is believed to be because of random errors that accumulate due to the back and forward propagation through the artificial neural network, as the velocity divergence field shows random noise patterns. Nevertheless, the divergence-free basis gives velocity fields that are roughly the same as the one where no divergence-free basis is used. However, the variations for the case with the divergence-free basis employed still yield a velocity field that is divergence free, which is preferred. Especially because the errors in the momentum equations are roughly the same. The reason why the absolute error in the TG case is lower than for the ISF is that the underlying velocity field, as well as the domain, are smaller in size/magnitude which therefore scales

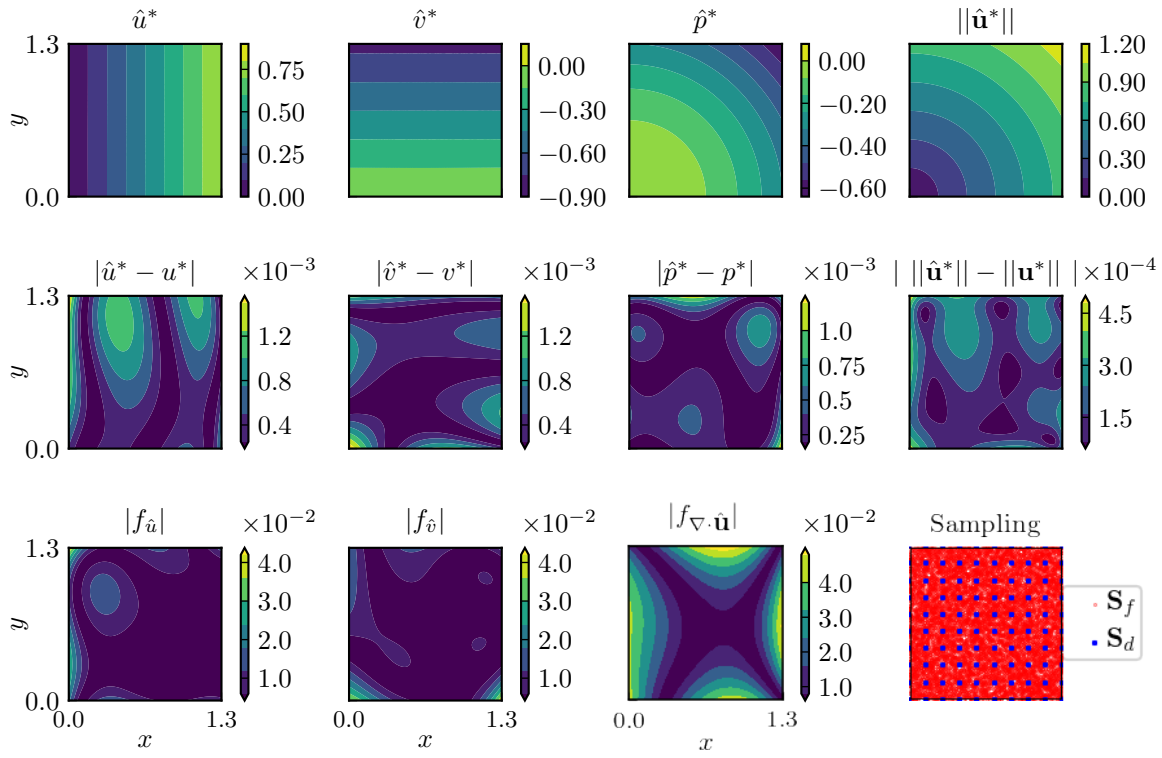


Figure 5.1: DPFINN solutions for the ISF case, without divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

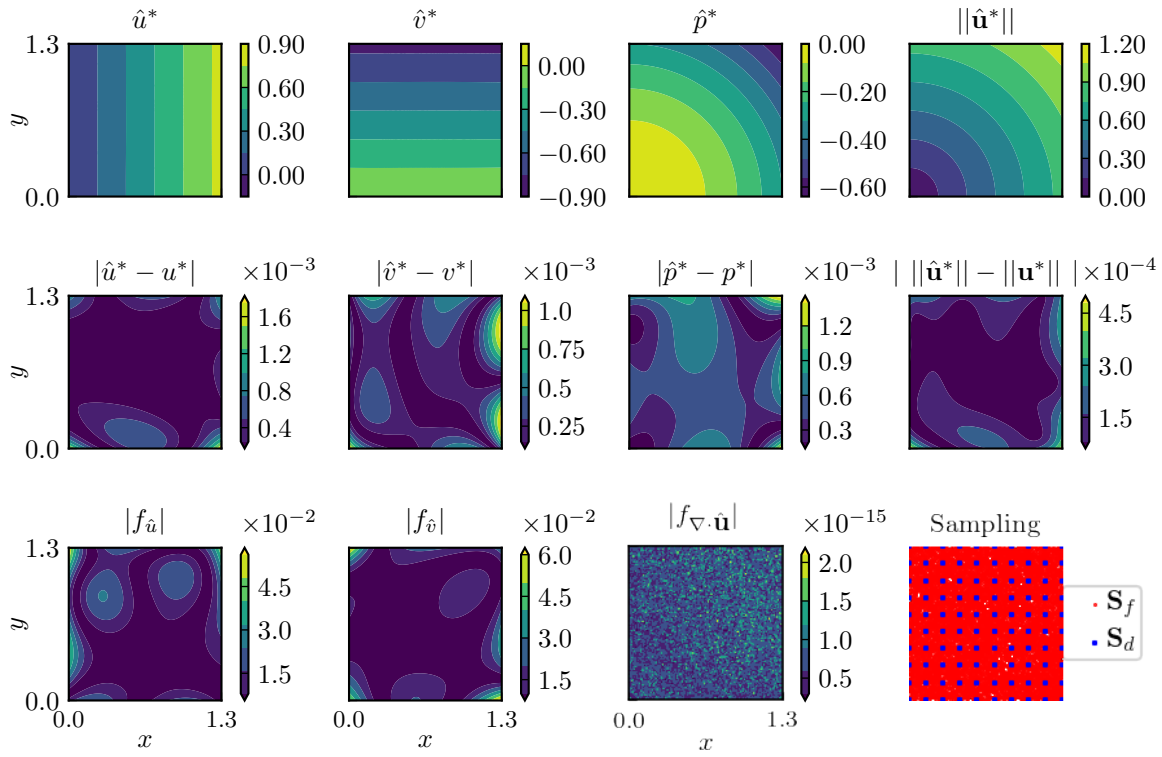


Figure 5.2: DPFINN solutions for the ISF case, with divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

the momentum equation magnitudes to a smaller one.

5.1.3 Hiemenz viscous stagnation flow

Since this case is viscous and therefore more complicated, a larger artificial neural network is used than for the previous cases (ISF and TG): $N_n = 20$, $La = 5$. However, this artificial neural network is still considered to be trainable, thus not too big in complexity. The results for the run without divergence-free basis are shown in Figure 5.5 while the results for this case with divergence-free basis is shown in Figure 5.6. Again, the flow fields are shown in the first rows, the discrepancies of these in the second row and the discrepancies of the Navier-Stokes equations (in *NN*-form) in the bottom rows. Inspection of the $|f_{\nabla\cdot\hat{\mathbf{u}}}|$ shows that the divergence of the velocity field is of the order $O(10^{-1})$ for the run without the divergence-free basis, while for the run with the divergence-free basis, the order is down to the resolution of the floating-point type: $O(10^{-15})$. Looking at the discrepancy of the flow variables, it becomes apparent that for the case without the divergence-free basis, there are distinct horizontal lines through the domains, especially for $|\hat{v}^* - v^{\hat{*}}|$. This is because the outputs of the artificial neural network are the flow variables itself. Unfortunately, the artificial neural network is too complex to show how the artificial neural network exactly forms these. However, an attempt can be made with a very small artificial neural network: $N_n = 4$, $La = 2$. The solutions are shown in Figure 5.7. The flow variable discrepancies are one order of magnitude higher than for the $N_n = 20$, $La = 5$ artificial neural network used, nevertheless, there are still distinct lines across the domain for which especially the discrepancy $|\hat{v}^* - v^*|$ exhibits the same characteristics. The network with the outputs of each neuron in each hidden layer is shown in Figure 5.8. It can be seen that the outputs of the second hidden layer have horizontal characteristics i.e. the outputs vary mostly with $\hat{y}^{\hat{*}}$ except for the output of the second neuron in the second hidden layer. As that neuron exhibits features of the resulting pressure field $\hat{p}^{\hat{*}}$.

5.1.4 Lid-driven Cavity Flow

For this case, the troublesome areas are the top corners (as also discussed in subsection 4.4.4). The results from the DPFINN framework, without the divergence-free basis, are shown in Figure 5.9. Here the artificial neural network is $N_n = 10$, $La = 3$, $N_d = 10^2$ and $N_f = 10^2$ while the Navier-Stokes equations are used in *NN*-form. The maximum discrepancies are much higher than for the other test cases. These discrepancies are situated at the corners of the domain, as expected. These maximum discrepancies originate at the corners for all the flow variables as well as for the momentum equations. Throughout most of the domain, there is very little difference between the solutions for the flow variables. Both the framework with the divergence-free basis and the framework without it can infer the flow variables well. Especially the profiles for $\hat{u}^*$ and $\hat{v}^*$ along the center profiles are considered to be accurate and match the OpenFOAM solution closely (see Figure 5.11). The pressure $\hat{p}^*$ shows some larger disagreement but is still in relatively good agreement. Looking more closely at the corners of the domain (indicated in red), it shows that both of the frameworks have difficulty inferring the discontinuity of $\hat{u}^*$ at the lid of the domain. This can also be seen in the momentum profiles along the lid (shown in the bottom row), as near the corners, the momentum equations show large inaccuracies, which can also be seen in the contour plots in Figures 5.9. However, the divergence-free basis decreases these discrepancies slightly. This cannot be said for $\hat{v}^*$ along the left and right boundaries. The discrepancy with respect to the exact solution shows large spikes close to the corners, which are larger for the DPFINN framework with a divergence-free basis. The reason for this is that the large gradients of $\hat{u}^*$ with respect to x near the corner points due to the discontinuity need to be opposed by the gradient of $\hat{v}^*$ with respect to y but in the opposite sign, otherwise, the velocity at this location cannot be divergence-free. Therefore, this is not an issue of the divergence-free basis, which does what it is supposed to do but more of an issue of the test case itself as the discontinuity is perhaps not fully physically consistent. Lastly, the pressure $\hat{p}^*$ on the bottom of the domain looks like it deviates significantly more with respect to the other locations/profiles, however, the pressure varies very slightly on the bottom boundary and therefore the error is roughly similar to the other locations.

5.2 Effect of artificial neural network complexity

By increasing the artificial neural network complexity, it is hypothesized that the relatively more complex artificial neural networks are always better capable of inferring the flow variables accurately while the gradients of the flow variables can also be represented more accurately. Several architectures are

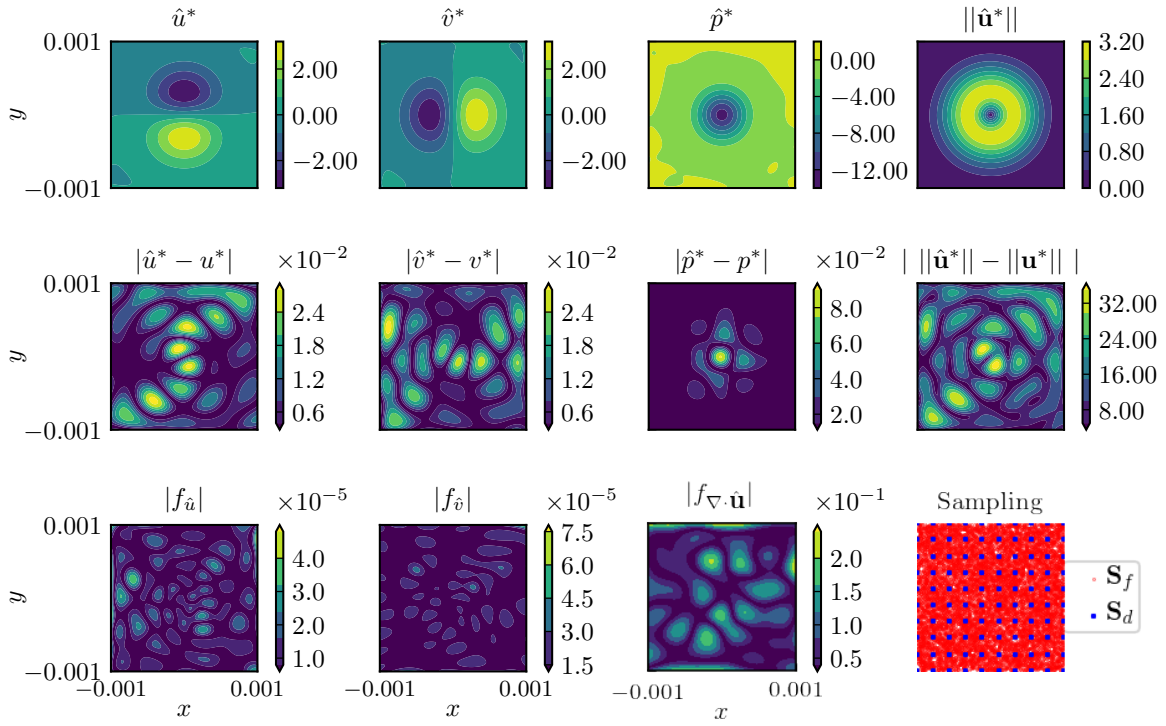


Figure 5.3: DPFINN solutions for the TG case, without divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

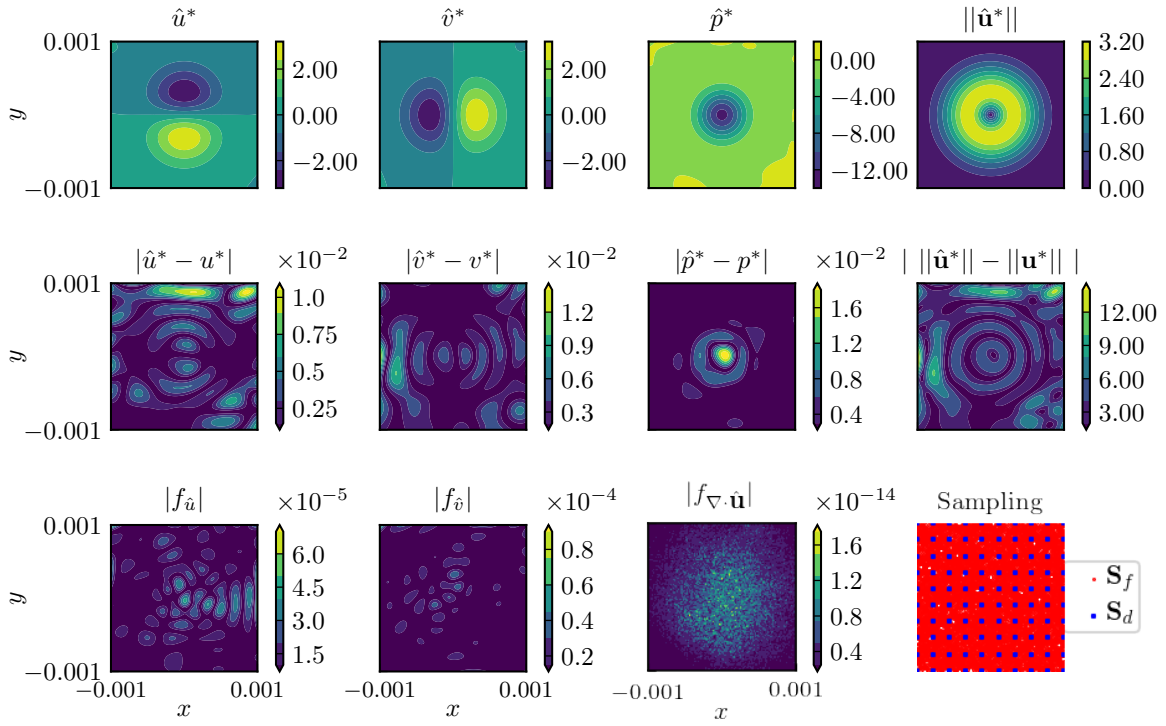


Figure 5.4: DPFINN solutions for the TG case, with divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

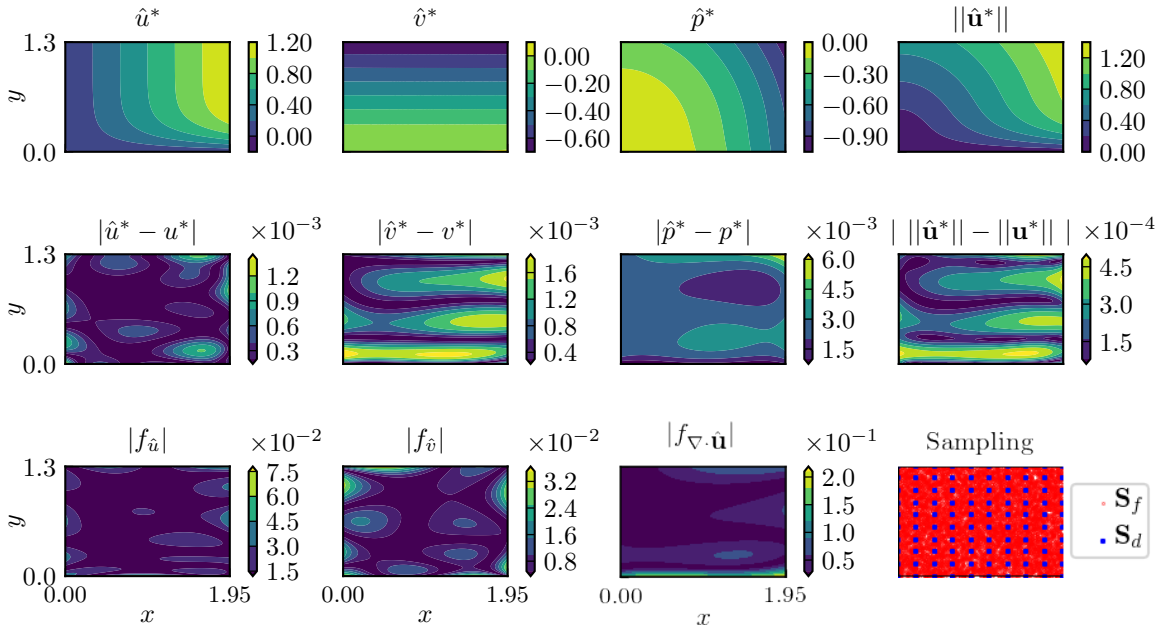


Figure 5.5: DPFINN solutions for the HM case, without divergence-free basis and an artificial neural network with $N_n = 20$, $La = 5$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Navier-Stokes equations in *NN*-form.

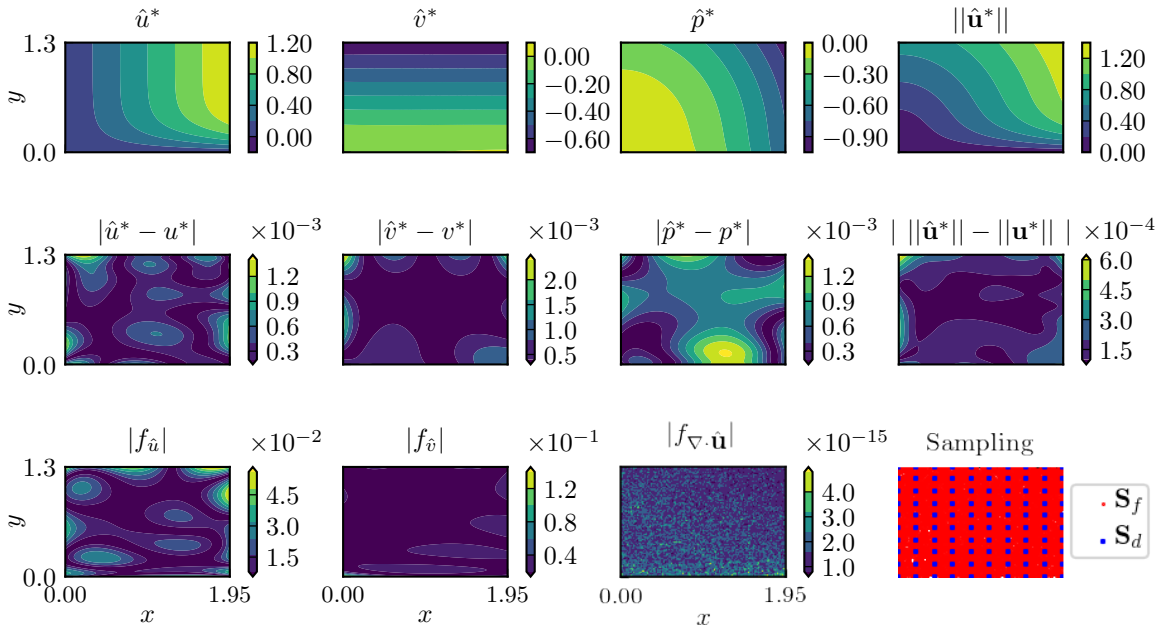


Figure 5.6: DPFINN solutions for the HM case, without divergence-free basis and an artificial neural network with $N_n = 20$, $La = 5$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Navier-Stokes equations in *NN*-form.

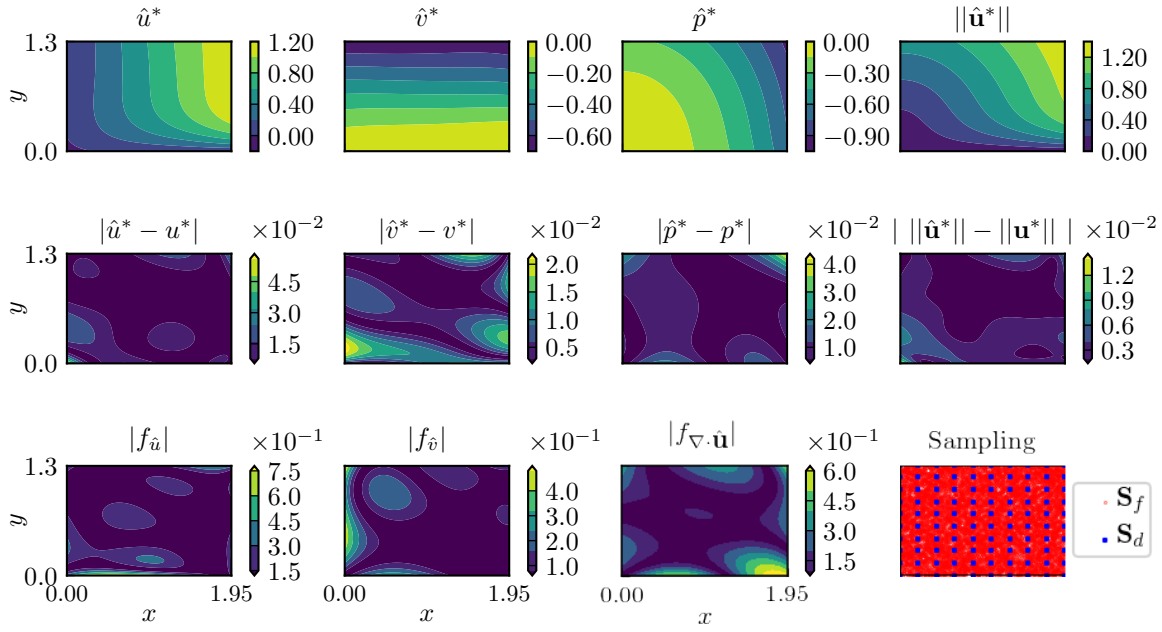


Figure 5.7: DPFINN solutions for the HM case, without divergence-free basis and an artificial neural network with $N_n = 4$, $La = 2$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Navier-Stokes equations in *NN*-form.

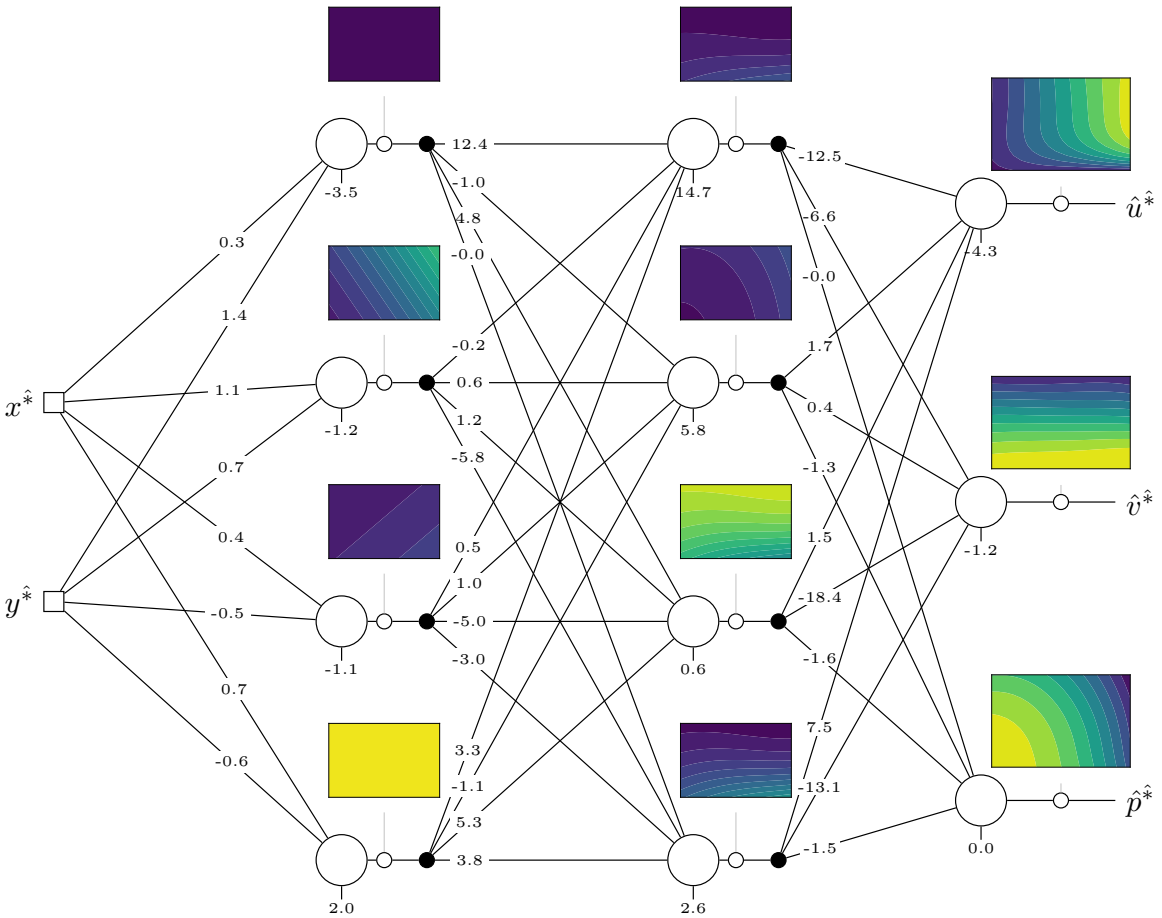


Figure 5.8: Network inputs and outputs for DPFINN solution of the HM case, without divergence-free basis and an artificial neural network with $N_n = 4$, $La = 2$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Navier-Stokes equations in *NN*-form.

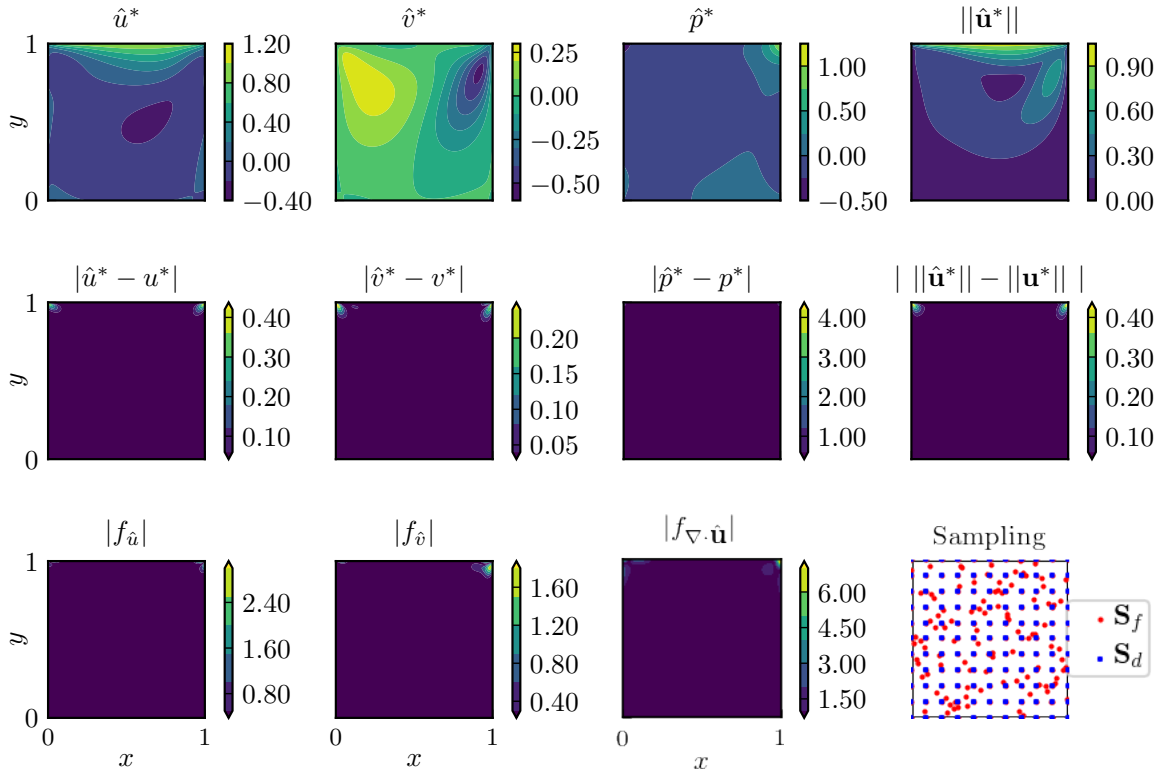


Figure 5.9: DPFINN solutions for the LDC case, without divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 11^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE loss function used for both data and physics and Navier-Stokes equations in NN-form.

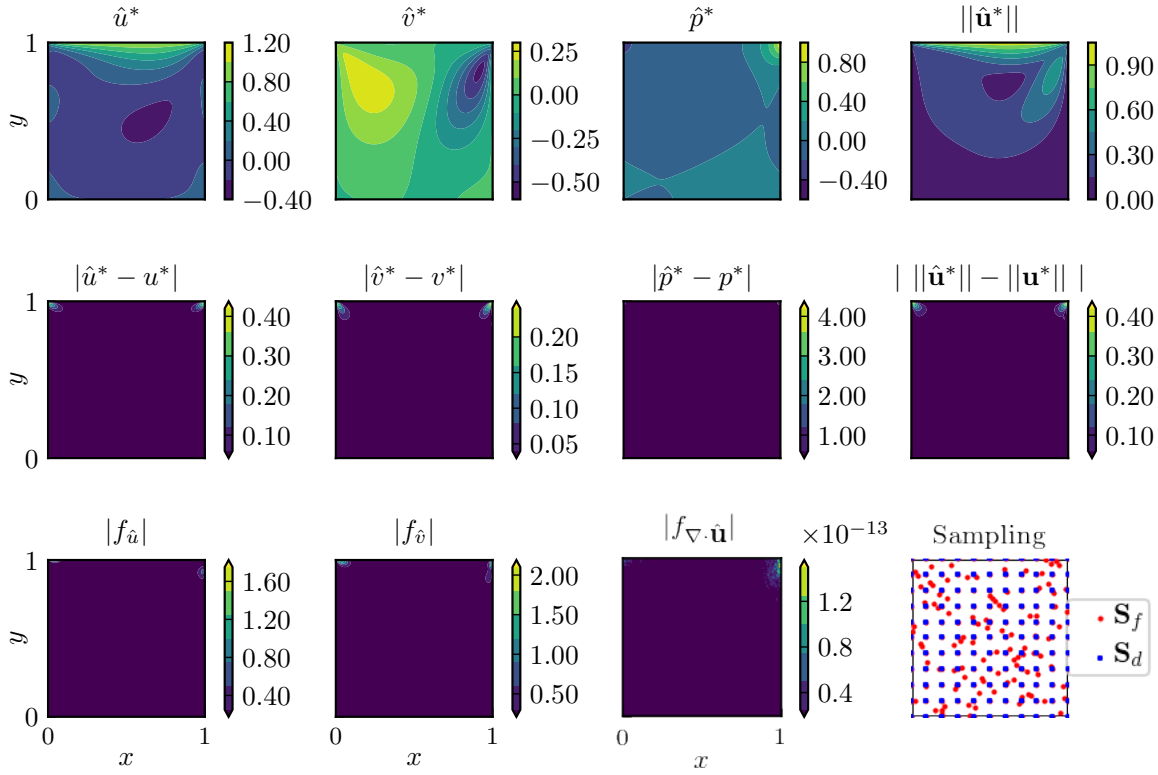


Figure 5.10: DPFINN solutions for the LDC case, without divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 11^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Navier-Stokes equations in NN-form.

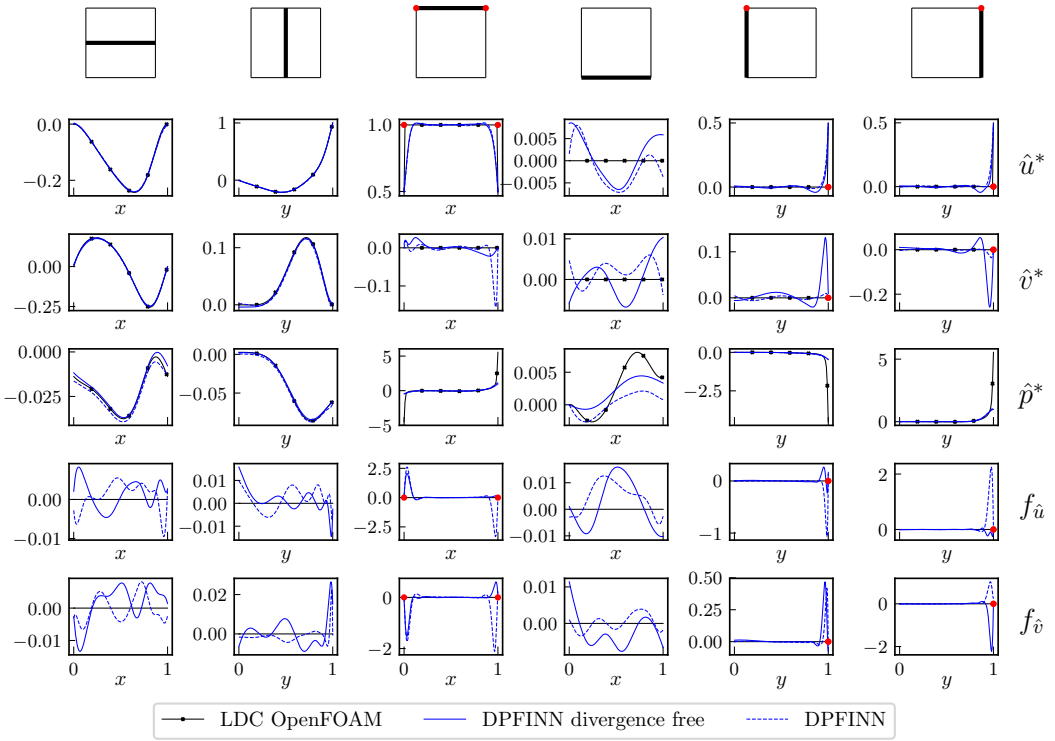


Figure 5.11: DPFINN profiles for LDC case with and without divergence-free basis employed. artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 11^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE loss function used for both data and physics and Navier-Stokes equations in *NN*-form. Red dots indicate areas of discussion emphasis.

tested for the test cases and are described in the following subsections. For all the runs shown here, the Euler and Navier-Stokes equations are in *NN*-form. Furthermore, the MSE loss function type is used for both data and physics and $N_d = 10^2$ (uniformly distributed), $N_f = 100^2$. Moreover, the MSE loss function type is used for both the data and physics loss functions for the results in this section.

5.2.1 Inviscid Stagnation Flow

For the inviscid stagnation flow case, three architectures are considered, the first one being very simple: just one hidden layer and four artificial neurons ($N_n = 4$, $La = 1$). The reason for this network is because the velocity fields are linear profiles in x or y, therefore, a very simple artificial neural network would in principle be able to approximate these profiles. The second architecture is the same as already used for the previous section: $N_n = 10$, $La = 3$. The third one is a lot more complex: $N_n = 50$, $La = 10$, which is most likely not necessary but is included to analyze whether a very complex network would improve the momentum equations. The profiles for all inferred the flow variables ($\hat{u}$,$\hat{v}$,$\hat{p}$) as well as the momentum equations ($f_{\hat{u}}$,$f_{\hat{v}}$) are shown in Figure 5.12. Looking at the profiles for $\hat{u}$ along the horizontal profiles, it is clear that all the artificial neural networks, including the smallest one ($N_n = 4$, $La = 1$) are able to reproduce the linear profiles. The same is the case for $\hat{v}$ but then along the vertical profiles. Moreover, the pressure $\hat{p}$ is also inferred properly and matches the exact solutions well, which is remarkable, since the complexity for the smallest network is very low. Nevertheless, for the profiles where either $\hat{u}$ is constant (being the vertical ones) and where $\hat{v}$ is constant (being the horizontal ones), the more complex networks perform slightly better, however not much. The smallest network produces a parabolic profile, which is the solution that gives the lowest mean discrepancy throughout the domain. This corresponds to the loss function choice, as the MSE loss function type is used for the data loss function. The momentum equations are plotted in the bottom two rows of the profiles. They show that indeed the more complex networks have momentum equation discrepancy that oscillates more but is not significantly closer to zero than the one for the $N_n = 4$, $La = 1$ artificial neural network. Nevertheless, the highest momentum discrepancies are obtained on the boundaries of the domain.

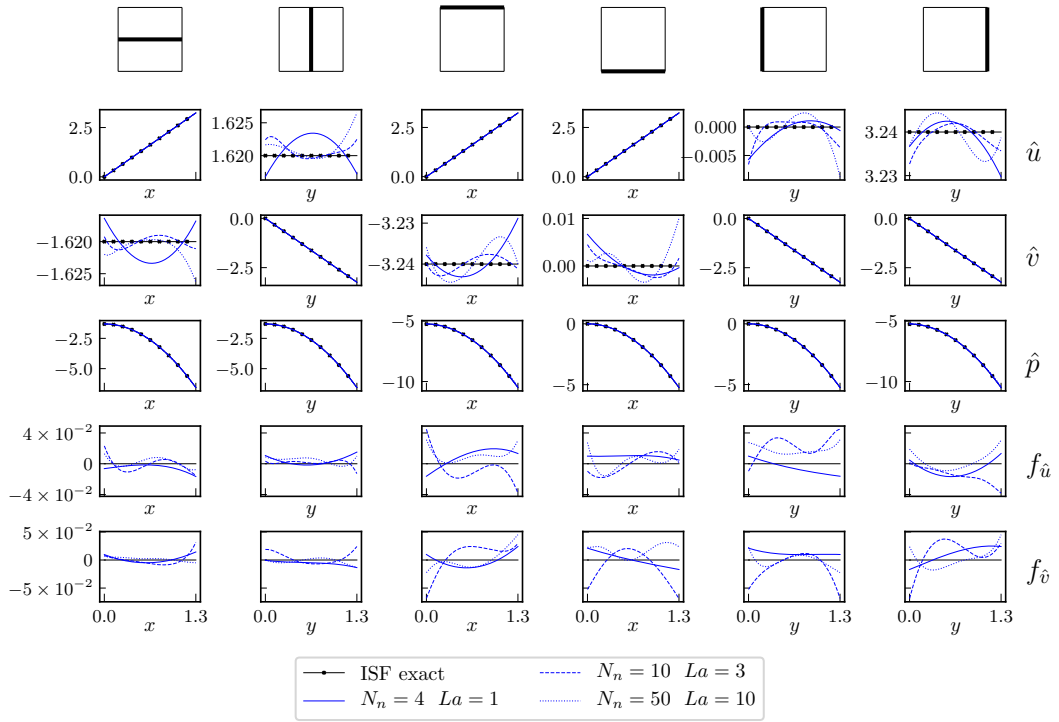


Figure 5.12: DPFINN divergence-free profiles for ISF case with varying artificial neural networks. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

5.2.2 Taylor-Green vortex

For the Taylor-Green vortex case, it is expected that slightly more complex artificial neural networks are required. This, since the velocity profiles, do not exhibit linear features even though they are smooth. Nevertheless, a small artificial neural network $N_n = 4$, $La = 2$ is used as well as an artificial neural network with size $N_n = 10$, $La = 3$. The solution profiles are shown in Figure 5.13. The smaller artificial neural network does not perform well, while a similar small artificial neural network did work reasonably well for the ISF case. However, the $N_n = 10$, $La = 3$ artificial neural network works well. It can infer the profiles for the flow variables well and the momentum discrepancies are also reasonably low. For the $N_n = 10$, $La = 3$ artificial neural network run, the effect of the divergence-free basis can be observed. By looking at the slope of the $\frac{\partial \hat{u}^*}{\partial x^*}$ and $\frac{\partial \hat{v}^*}{\partial y^*}$ in the middle of the domain (at the indicated red dot), it can be seen that $\frac{\partial \hat{u}^*}{\partial x^*}$ is positive here. While $\frac{\partial \hat{v}^*}{\partial y^*}$ is negative but with the same magnitude. This confirms that the velocity field is divergence-free even though it does not match the exact solution.

5.2.3 Hiemenz Stagnation Flow

For the viscous Stagnation flow, it was already established in Section 5.1 that the $N_n = 20$, $La = 5$ artificial neural network produced good results for the inference of the flow variables (refer to Figure 5.6 again). For the inviscid case, the simple artificial neural network $N_n = 4$ $La = 1$ was able to capture the profiles for the velocity fields surprisingly well. Here, that same artificial neural network is taken but with one extra hidden layer, since a viscous and therefore more complicated case is considered here. The artificial neural network is trained for this case and the results for both artificial neural networks are shown in Figure 5.14. The best results are unsurprisingly obtained by the most complex artificial neural network. Nevertheless, the smaller artificial neural network is able to find the profiles for the flow variables reasonably well, while not as good as the more complex artificial neural network. A notable difference for $\hat{p}^*$ is found at the top right corner of the domain (red dot). Here, the more complex artificial neural network provides more capability of inferring the pressure accurately. It is worth noting that the profiles for the momentum equations $|f_{\hat{u}}|$ and $|f_{\hat{v}}|$ show that the discrepancies are closer to zero for the larger artificial neural network than for the smaller one, as expected. However, for both (and all the other runs for this case) the no-slip velocity condition at the wall (the bottom of the domain),

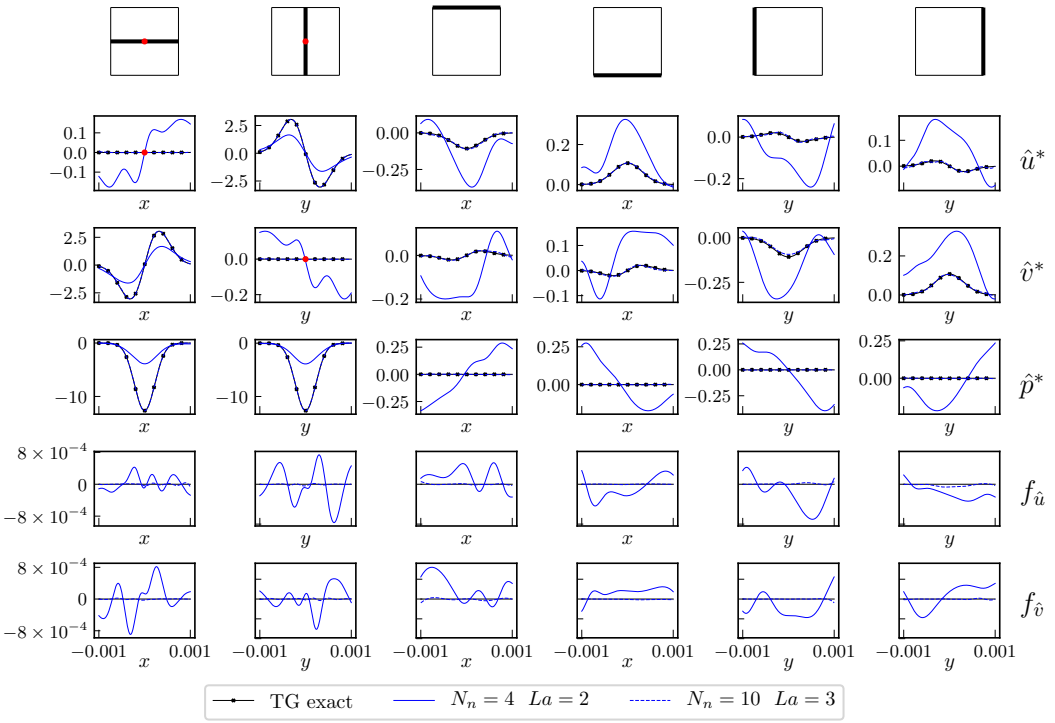


Figure 5.13: DPFINN divergence-free profiles for TG case with varying artificial neural networks. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

is not satisfied completely. For both $\hat{u}^*$ and $\hat{v}^*$, still, variations are observed.

5.2.4 Lid Driven Cavity Flow

As already mentioned, the corners of the lid-driven cavity flow form issues, as continuities in the u-velocity exist there. Results for an artificial neural network with $N_n = 10$ and $La = 3$ were already given, see Figure 5.11. Here, it was shown that the $N_n = 10$ and $La = 3$ artificial neural network produced reasonable results but showed considerate deviation at the corner points. Next to that artificial neural network, a more complex one is taken: $N_n = 20$, $La = 5$ for comparison purposes. The added layers are expected to improve the predictions over the entire domain slightly. The results are shown in Figure 5.15. From the Figure, two conclusions can be drawn from the more complex artificial neural network. Overall, the inference accuracy of the flow variables does not increase much, if hardly at all. There are little changes for the $\hat{u}^*$ and $\hat{v}^*$ velocities while the $\hat{p}^*$ along the horizontal centerline and the bottom the domain is deemed to be slightly worse. Secondly, the momentum equation profiles of $f_{\hat{u}}$ and $f_{\hat{v}}$ show spikes that are larger than for the less complex case. Therefore, the more complex artificial neural network is concluded to not increase the accuracy with respect to the less complex artificial neural network significantly.

5.3 Effect of sampling

For the results shown so far the amount of velocity field samples that were used was $N_d = 10^2$ (uniformly distributed) while the physics collocation $N_f = 100^2$. Naturally, it is of interest to vary the amount of sampling. This, to investigate what the minimum amount of velocity field data is in order for the framework to effectively predict the flow variables across the domain(s). But also, whether the added physics collocation points can increase the accuracy of the inferred velocity fields throughout the domain(s), in case the number of data points proved insufficient. For the results in this section, again the MSE loss function type is used for both the data and physics loss function. Moreover, for all the runs shown here, the Euler and Navier-Stokes equations are taken in *NN*-form.

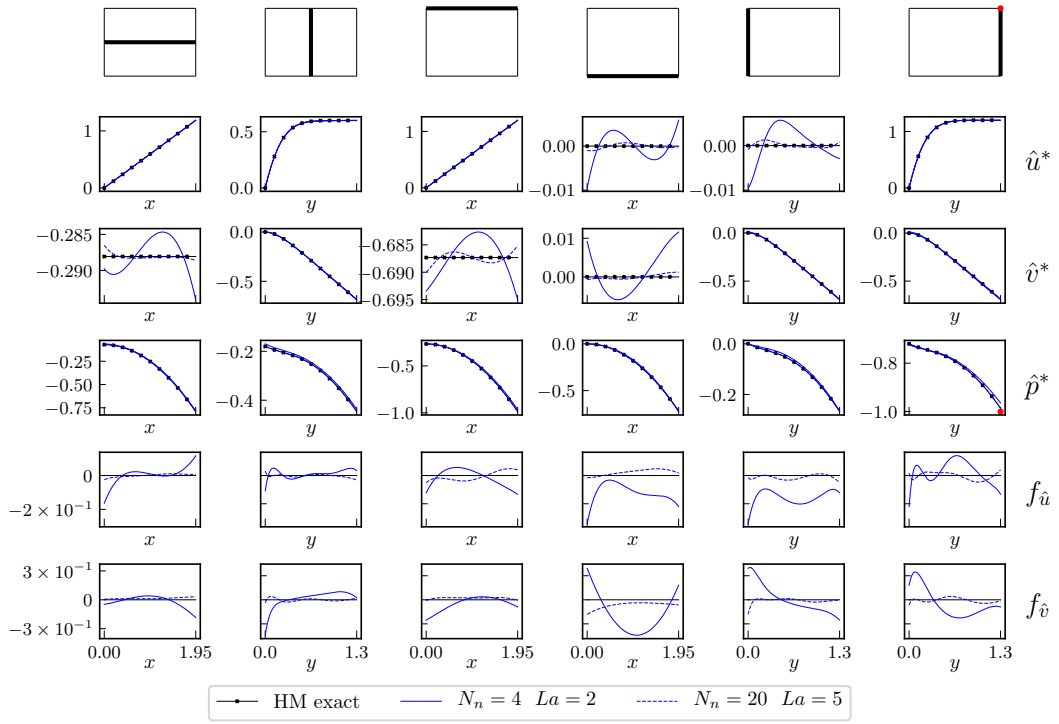


Figure 5.14: DPFINN divergence-free profiles for HM case with varying artificial neural networks. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Navier-Stokes equations in *NN*-form.

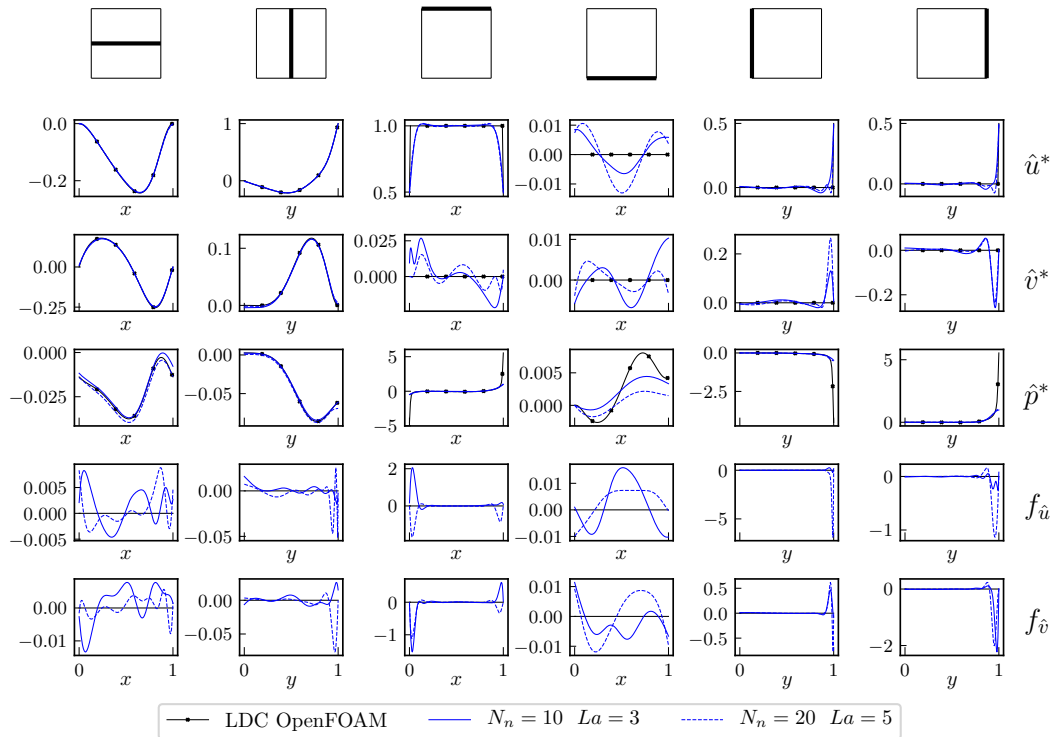


Figure 5.15: DPFINN divergence-free profiles for LDC case with varying artificial neural networks. Data sampling is $N_d = 11^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE loss function used for both data and physics and Navier-Stokes equations in *NN*-form.

5.3.1 Inviscid Stagnation Flow

First, the inviscid stagnation flow test case is taken. Since this flow case has only velocity fields that vary with x and y, in principle, using only data points at the corner points would provide enough information to the framework. The data points used are only at the corners, which totals to four data points ($N_d = 4$). The artificial neural network taken is one with $N_n = 10$, $La = 3$, since in the architecture comparison this artificial neural network gave good results with $N_d = 10^2$ (see Figure 5.12). Here, the most complex artificial neural network $N_n = 50$, $La = 10$ did not give significantly better results which is not expected here either. The results for this case, for a $N_n = 10$, $La = 3$ artificial neural network with $N_d = 2^2$ and $N_f = 10^2$ is shown in Figure 5.16. The velocity at the corners of the domain is approximated well, whereas, throughout the rest of the domain, the output of the framework deviates from the exact solution. This is not because the artificial neural network is not complex enough or because the choice tanh activation is not suitable. It is already established that this artificial neural network is able to infer the flow variables well. The discrepancies are purely due to the lack of information provided to the framework, it could be said that the problem is therefore ill-posed. The largest deviation of the pressure $\hat{p}^*$ is found at the top right corner. The reason for this is because the incompressible pressure is defined up to a constant, therefore the pressure is shifted to match the zero stagnation pressure at the bottom left of the domain. The bottom right plot shows the data points as well as the additional physics collocation points. To test whether increasing N_f would improve the predictions, N_f is increased to $N_f = 100^2$, still randomly sampled. The results are shown in Figure 5.17. The accuracy plots in the second row show that the discrepancies have increased slightly, this is also true for the pressure where the maximum deviation is still in the top right corner but increased by a factor 2.5 approximately. For the momentum discrepancies: they seemed to be decreased in the middle of the domain, but more spread out towards the edges. This is the result of the MSE loss function possibly, a small number of large deviations are penalized less.

In comparison, the same architecture ($N_n = 10$, $La = 3$) as well as the same formulation is used but now for a more significant amount of data points: $N_d = 30^2$, while keeping $N_f = 100^2$. The profiles are shown in Figure 5.18. Unsurprisingly, the results flow variable inference performance increases significantly. The added data informs the framework more in the middle of the domain, thus, on average the fit to the data is much better. A similar argument can be made about the momentum equations, the momentum equations discrepancy is lowered, although to a lesser degree as would be expected with such an increase in data injection.

5.3.2 Taylor-Green vortex

In the preceding sections, this flow case was applied to the framework with data sampling $N_d = 10^2$, which is reasonably large. To compare the results with a lower amount of sampling, the test case is applied to the framework with $N_d = 4^2$, still uniformly sampled. The results are compared in the profiles in Figure 5.22. A clear difference can be found in the profiles for $\hat{u}^*$ and $\hat{v}^*$. Especially on the boundaries, the velocity discrepancies are minimized on the four datapoints that each boundary contains. Unfortunately, the velocity fields vary much in between those points. It can be concluded that the addition of the physics did not help in increasing the accuracy between the datapoints. For the pressure profiles $\hat{p}^*$, the pressure sink in the middle of the domain is not well resolved and is about half the magnitude when compared to the exact solution (indicated by the red dot). Nevertheless, it satisfies the momentum equations because the gradients of the velocity are smaller in magnitude, which causes a convective term to be smaller in magnitude. Which in turn gives a pressure gradient that is too small, as it balances the convective term, and this results in a pressure sink that is not deep enough. Nevertheless, it satisfies the momentum equation which can be concluded from looking at the momentum equation profiles. When looking at the larger sampling size run $N_d = 10^2$ there are some larger variations in the momentum equations but the velocity profiles are much better resolved, the reason for this is that the additional datapoints are added to the set of physics collocation points, thus a higher density of physics collocation points is the result.

However, it is not a given that adding more physics collocation points to the $N_d = 4^2$ case will improve the results significantly. To illustrate this, the contour plots for $N_d = 4^2$ and $N_f = 10^2$ is shown in Figure 5.20 while the results for the same amount of data points and $N_f = 100^2$ case are shown in Figure 5.20. The differences are negligible. In fact, the flow variable discrepancies are roughly of the same order, which is also the case for the momentum equation discrepancies. Therefore, it is concluded that the additional physics collocation points do not necessarily improve the predictions for this low amount

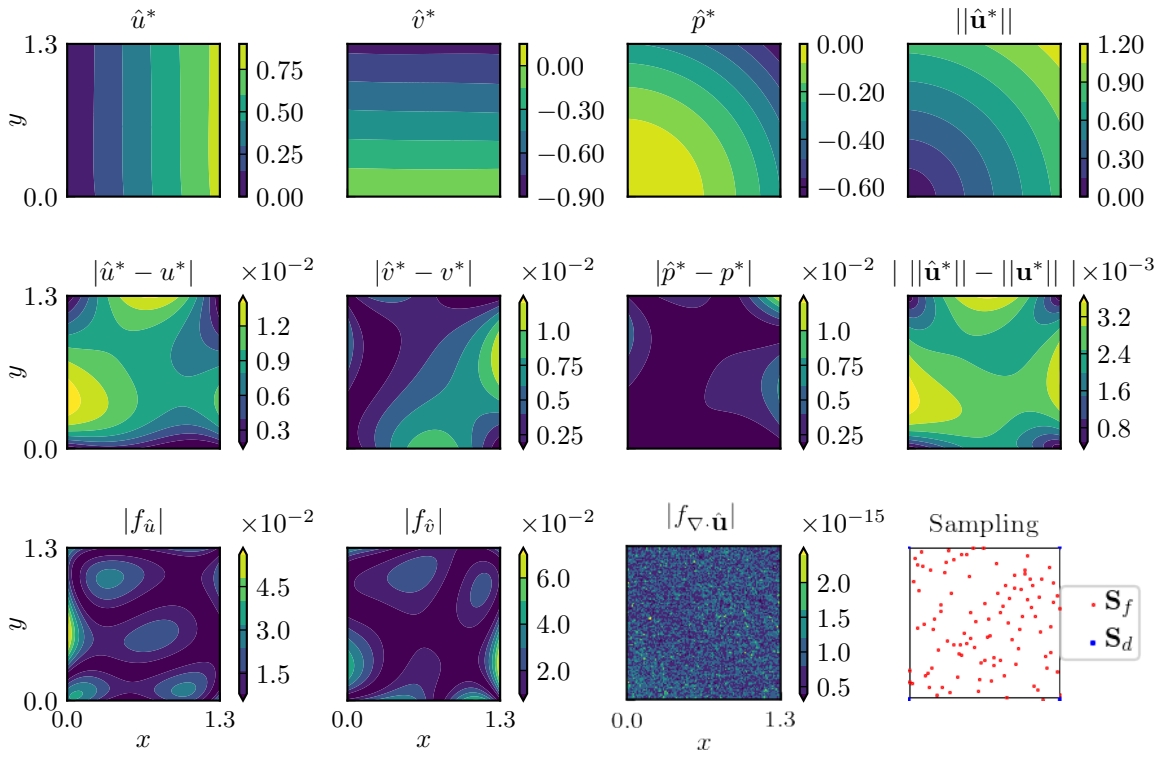


Figure 5.16: DPFINN solutions for the ISF case, with divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 2^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

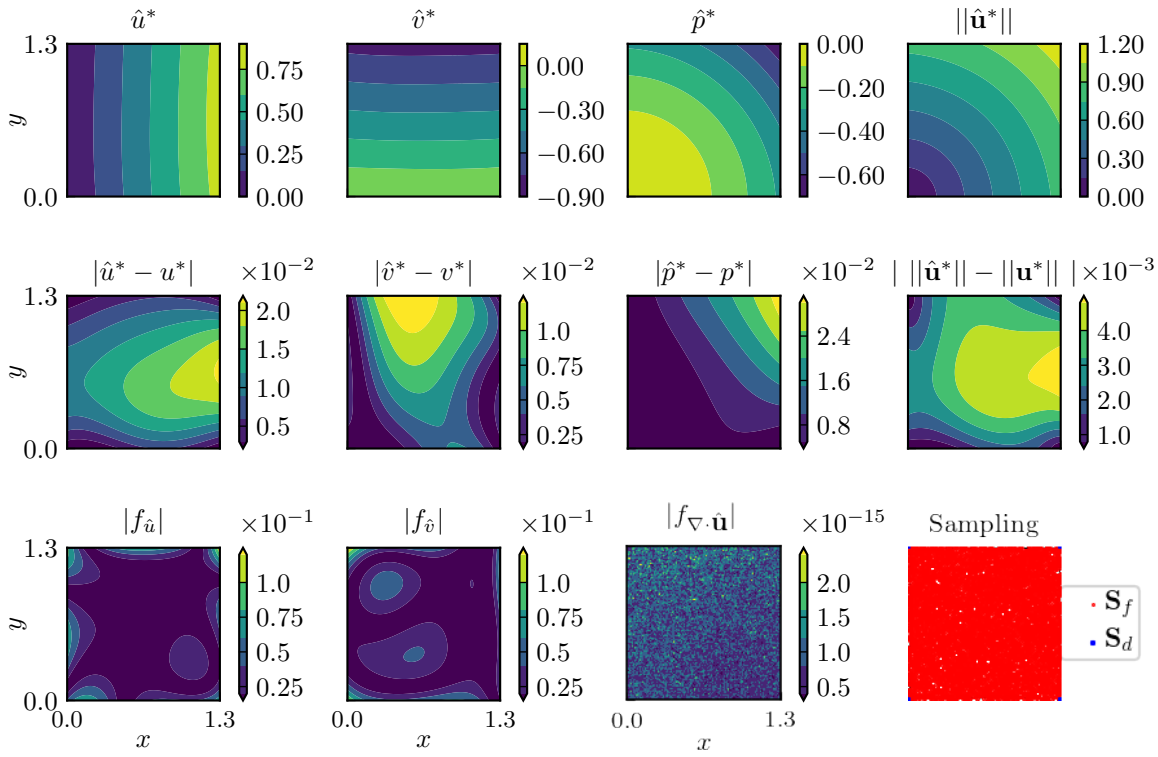


Figure 5.17: DPFINN solutions for the ISF case, with divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 2^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

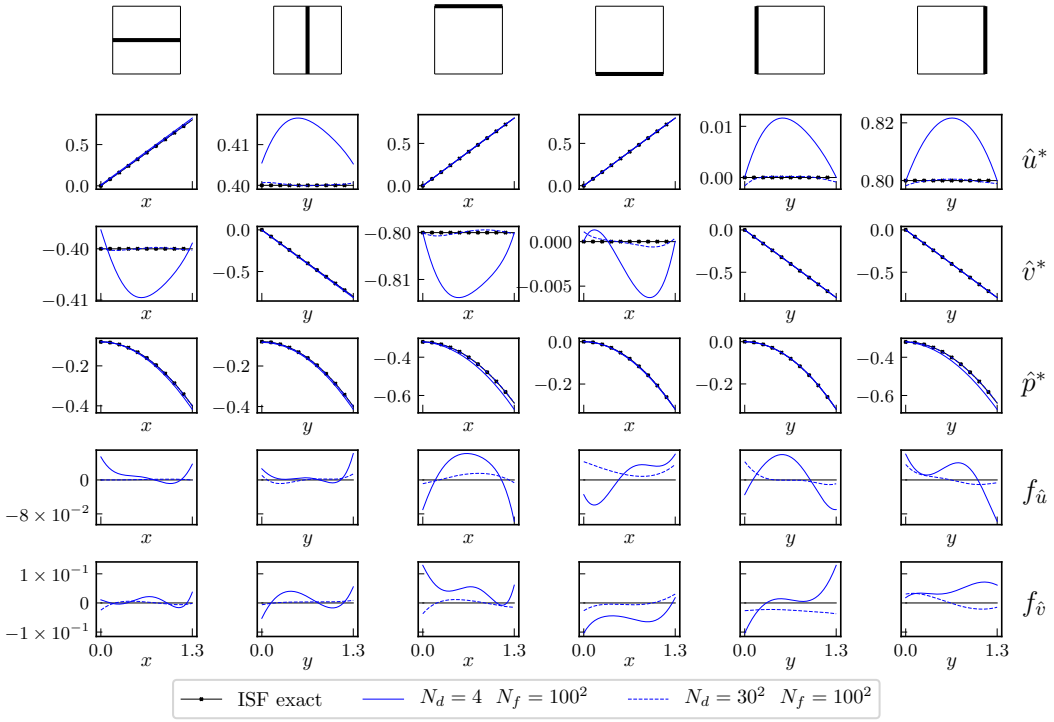


Figure 5.18: DPFINN divergence-free profiles for ISF case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 2^2$ and $N_d = 30^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

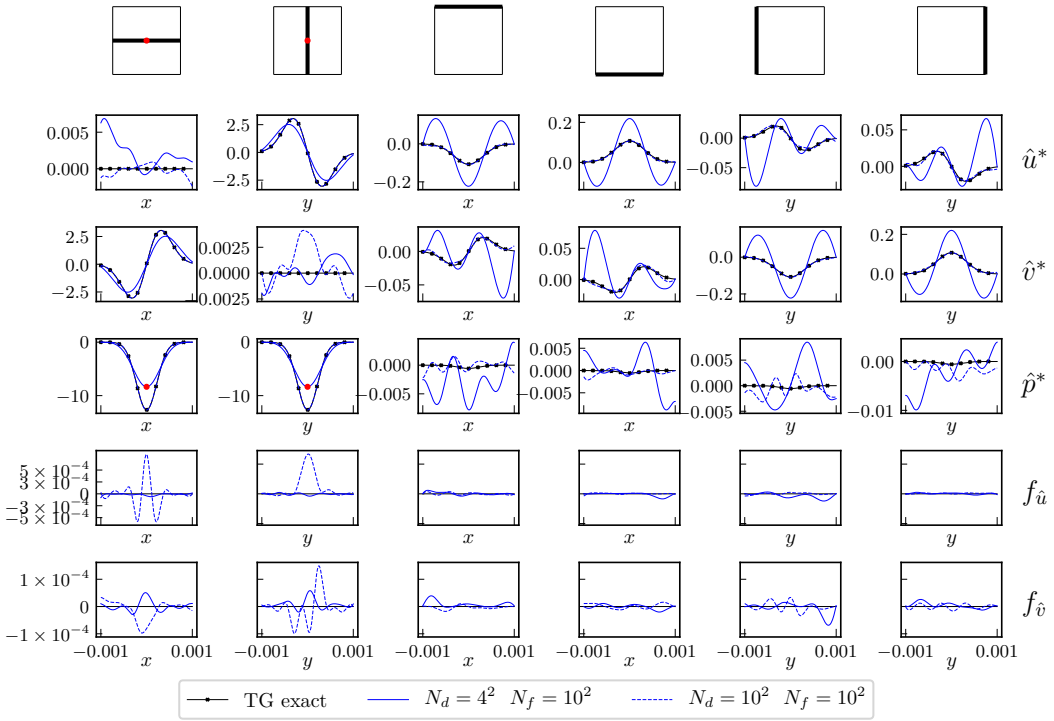


Figure 5.19: DPFINN divergence-free profiles for TG case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 4^2$ and $N_d = 10^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

of N_d, simply more data is required. This can probably be supplied by Dirichlet boundary conditions as well, which is the same as adding more data points there. However, due to the least-squares approach, there is no guarantee this will improve the accuracy of the velocity field predictions at the wall, without affecting/worsening the predictions of the framework for the rest of the domain.

5.3.3 Hiemenz Stagnation Flow

For the Hiemenz stagnation flow, a comparison can be made where the domain is sampled uniformly with $N_d = 4^2$ datapoints and $N_d = 20^2$. As shown in Figure 5.22, the results between the runs vary very little. It could be concluded that the addition of the smallest amount of data points was already sufficient to be informative for the framework. For the $N_d = 4^2$ case, there are four data points in each direction, therefore most of the boundary layer is sampled very poorly. Yet, the framework was able to find the correct boundary layer profile as well as the pressure field related to it.
To test the capabilities of the data injection for this case, the amount of data points is lowered to just three in each direction: $N_d = 3^2$, while maintaining the same amount of physics collocation points $N_f = 10^2$. The solution is showed in Figure 5.23. Remarkably, the framework is still able to infer the flow fields with relatively good accuracy. At least one that is comparable to other results obtained for the HM case. This means that for this case to be inferred reasonably well, only nine data points are required. The discrepancies for the velocity fields are highest in between the data points. The question is whether increasing the amount of additional physics collocation points decreases the discrepancies or increases the accuracy between the points. To test this, the amount of additional physics collocation points are increased to $N_f = 100^2$, and are shown in Figure 5.24. The maximum discrepancies in the velocity fields are halved while the discrepancy of the pressure is roughly the same but slightly more spread out. Moreover, the discrepancies are slightly more spread out while the maximum discrepancies of the momentum equations are pushed more towards the boundaries. Thus, to conclude, increasing physics collocation points here, increased the quality of the predictions slightly.

5.3.4 Lid Driven Cavity Flow

Inferring the flow fields and minimizing the physics could be a contradictory minimization goal that cannot be minimized both at the same time. This is tested for the lid-driven cavity flow. Here the domain is sampled with only four points in each direction which produces a domain sampled with $N_d = 4^2$ data points and for another run, the domain is sampled $N_d = 10^2$. The profiles are again shown in Figure 5.25. The extra data points result in the discontinuity of the lid velocity to be inferred better, yet not fully. Or in other words, most of $\hat{u}^*$ along the lid is inferred better for the $N_d = 10^2$ case. However, the spikes in the discrepancies for the momentum equations are slightly increased. This gives rise to the conclusion that the increased accuracy in the flow variables itself does not automatically improve the discrepancies of the gradients and therefore the momentum equations. At least locally, the profiles for the momentum equations suggest that a similar amount of total discrepancy is roughly the same, but just pushed more towards the corners of the lid. The discrepancies of the momentum equations throughout the rest of the domain are lowered.

5.4 Effect of physics loss function formulation

The formulation and complexity of the loss function is dependent on the type of loss function as well as the number of data points and collocation points used. Throughout the previous sections, the MSE loss function type was used solely, this section aims to present the results for all the loss function combinations. In total, there are four different combinations for the loss function formulation, as either MSE or SSE can be utilized for both the data and the physics loss function. The choice of the loss function provides relative scaling between the loss terms. When the SSE is used for one and MSE for the other, this means that more emphasis is placed on the first one. For all the previous analysis runs, the loss function combination that was used was MSE and MSE, therefore in principle, the loss function would be somewhat balanced, especially if the same amount of data and collocation points were used. However, altering the loss function to change the physics inclusion can either be done by changing the loss function type but also by changing the physics inclusion. Throughout the previous sections, the *NN*-form of the fluid flow equations were used. However, the *ND*-form and the *D*-form is also available. The following subsection aims to show the implications of the different physics inclusion options. However, the artificial neural network is fixed to the $N_n = 10$, $La = 3$ one.

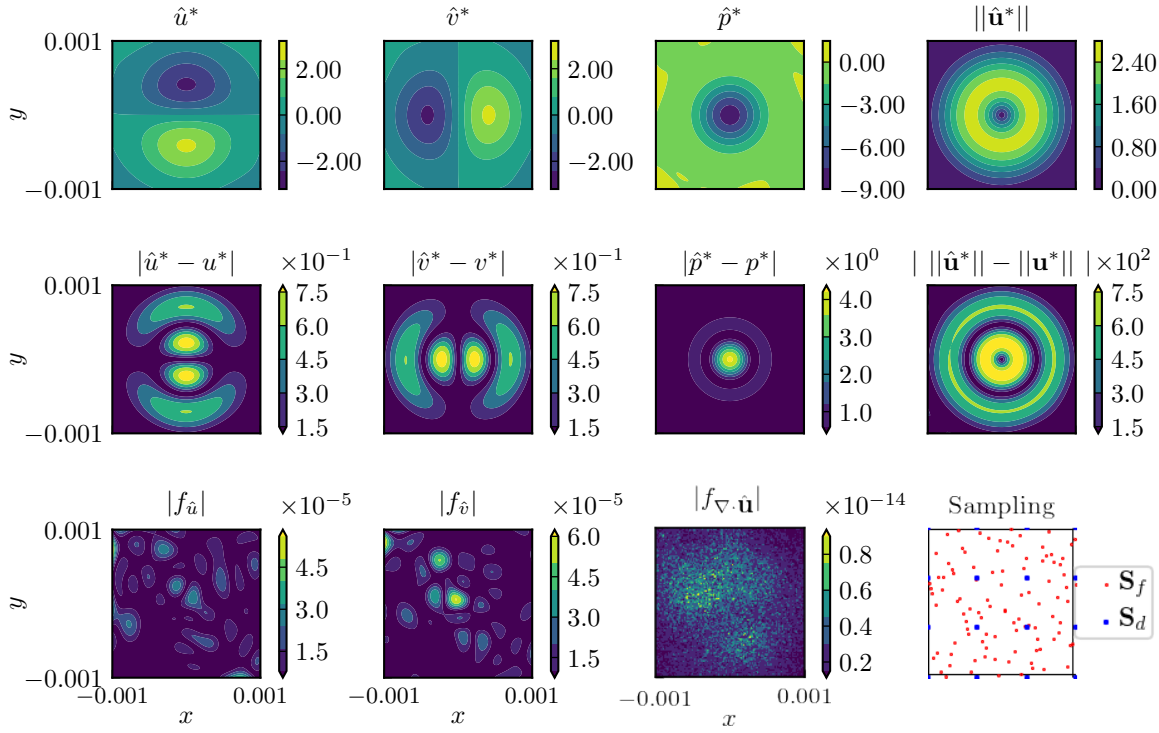


Figure 5.20: DPFINN solutions for the TG case, with divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 4^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

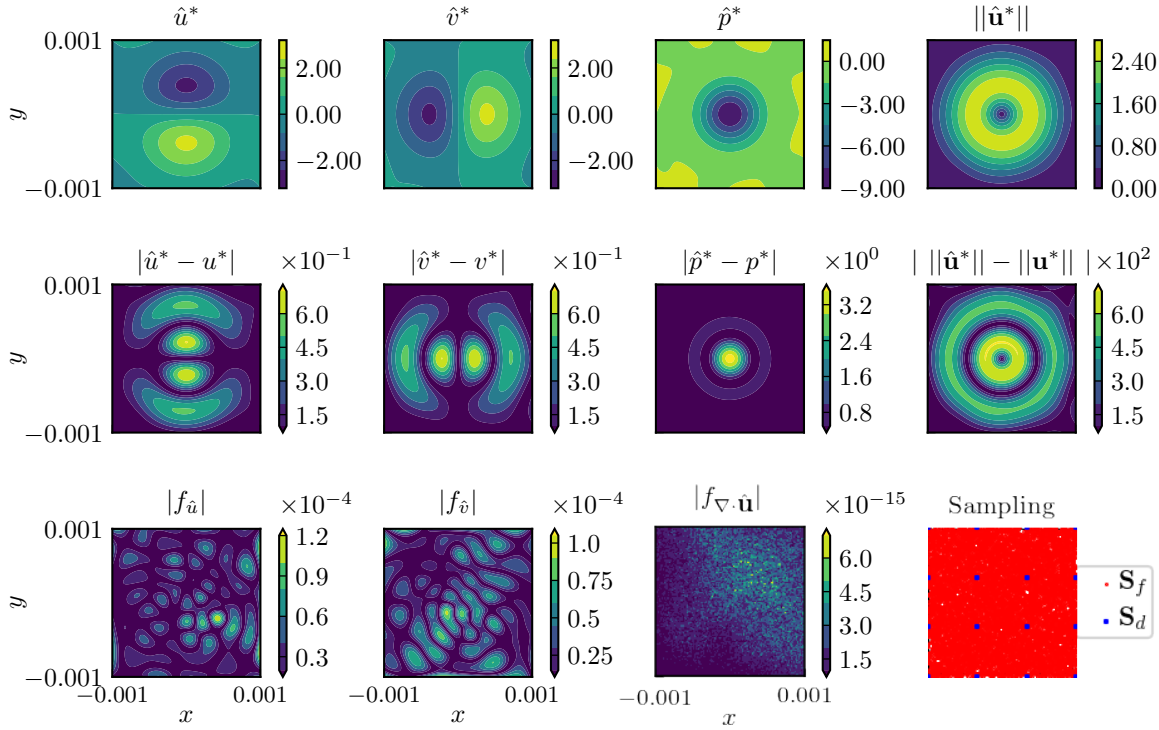


Figure 5.21: DPFINN solutions for the TG case, with divergence-free basis and an artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 4^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics and Euler equations in *NN*-form.

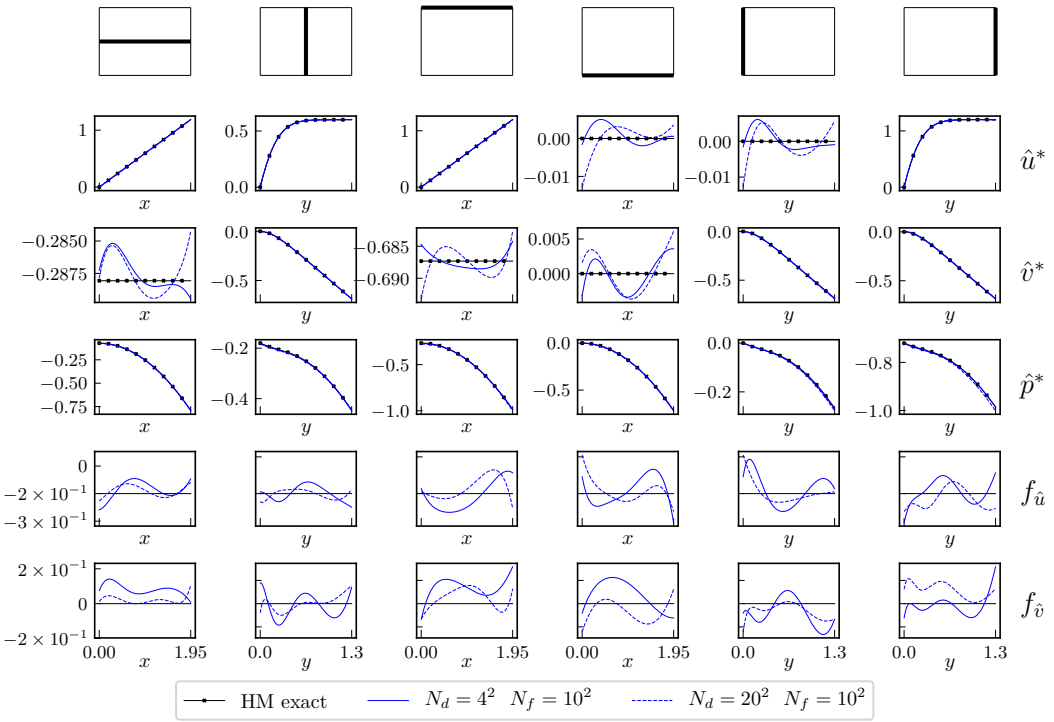


Figure 5.22: DPFINN divergence-free profiles for HM case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$ and $N_d = 20^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE loss function used for both data and physics. Navier-Stokes equations in *NN*-form.

5.4.1 Inviscid Stagnation Flow

The previous results for the ISF test case have shown that this case is fairly easily predicted by the various artificial neural networks. Here, the effect of different forms for the Euler equations are considered. For this, the artificial neural network: $N_n = 10$, $La = 3$ is used, with sampling $N_d = 10^2$ and $N_f = 100^2$. The results for the implementations of the Euler equations in *NN*-form, *ND*-form and *D*-form is shown in Figure 5.26. The variations of the flow variables with respect to each other and the exact solutions are believed to be negligible. This is not of a surprise since the difference between the data and the artificial neural network outputs in the data loss function are not changed, this is still carried out in by comparing $\hat{u}^*$ and $\hat{v}^*$ with respect to u_d^* and v_d^*. Nevertheless, the dimensional Euler equations (*D*-form) implemented in the loss function for the physics gives the lowest discrepancy for the momentum equations while the *NN*-form and *ND*-form yield similar results.

Next, another possibility is researched, which is the change of the loss function type for the physics from the MSE type to the SSE type. The profiles are shown in Figure 5.27. Similarly as the other Figures, the bottom two rows give the momentum equations $f_{\hat{u}}$, $f_{\hat{v}}$. What can be concluded from these profiles is that the SSE loss function significantly reduces the discrepancy of the momentum equations. Moreover, the inference quality of the rest of the flow fields did not decrease significantly.

5.4.2 Taylor-Green Vortex

In comparison to the ISF case, the results for changing the Euler equation form within the loss function is completely different. The TG case with $N_n = 10$, $La = 3$, $N_d = 10^2$ and $N_f = 100^2$ and Euler equations in *NN*-form, *ND*-form and *D*-form is shown in Figure 5.28. Here the effect of the forms is also more pronounced on the flow fields. The *D*-form gives the best accuracy for the velocity fields while the momentum equations discrepancy is highest in comparison to the other forms. Also, the *D*-form gives the highest discrepancies for the pressure field along the boundaries. The best result is believed to be found by the scaled version: the *NN*-form, since it gives the best quality predictions for both the velocity fields as well as the pressure field. Moreover, it shows the lowest discrepancy in the momentum equations along with the *ND*-form Euler equations. However, the *ND*-form does not infer the velocity fields well along the boundaries.

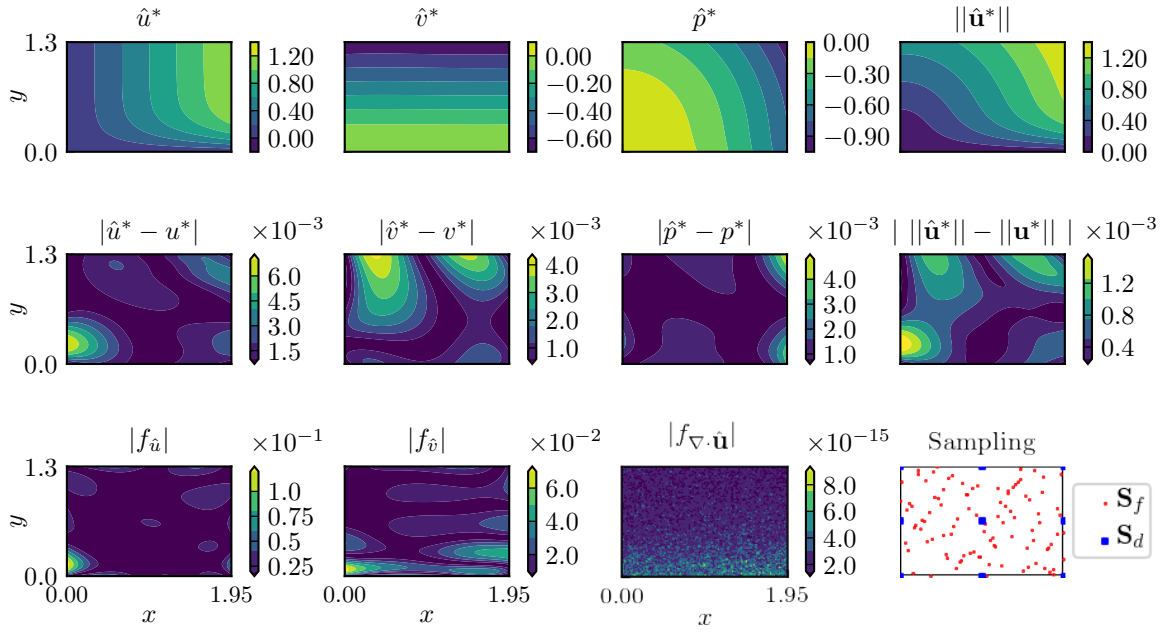


Figure 5.23: DPFINN solutions for the HM case. Artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 3^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE loss function used for both data and physics. Navier-Stokes equations in *NN*-form.

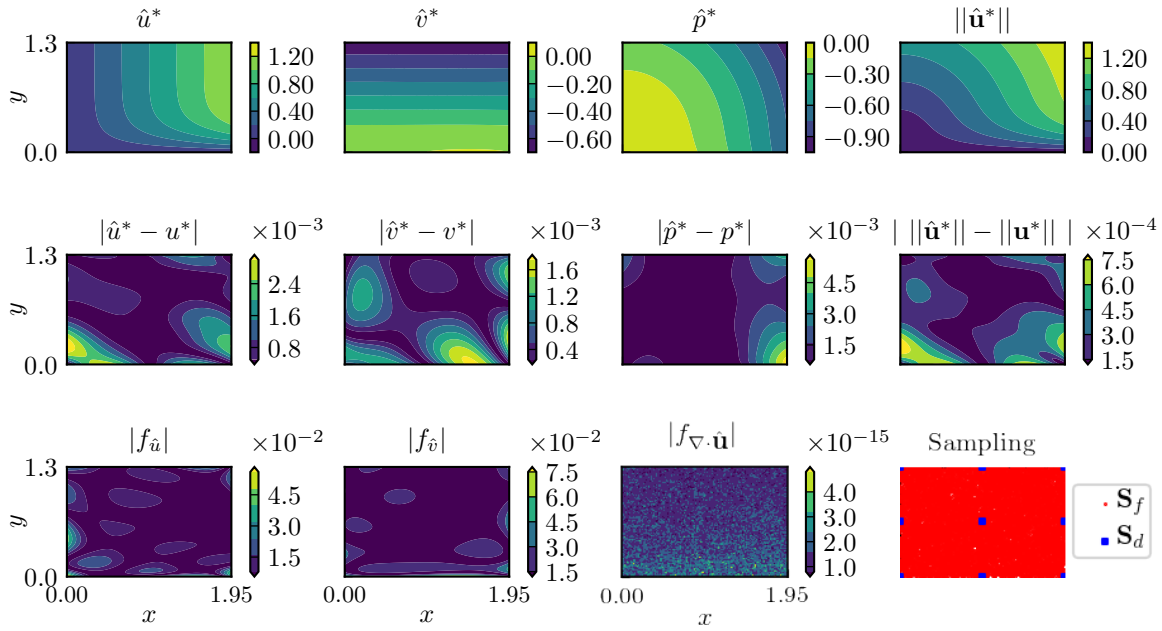


Figure 5.24: DPFINN solutions for the HM case. Artificial neural network with $N_n = 10$, $La = 3$. Data sampling is $N_d = 3^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE loss function used for both data and physics. Navier-Stokes equations in *NN*-form.

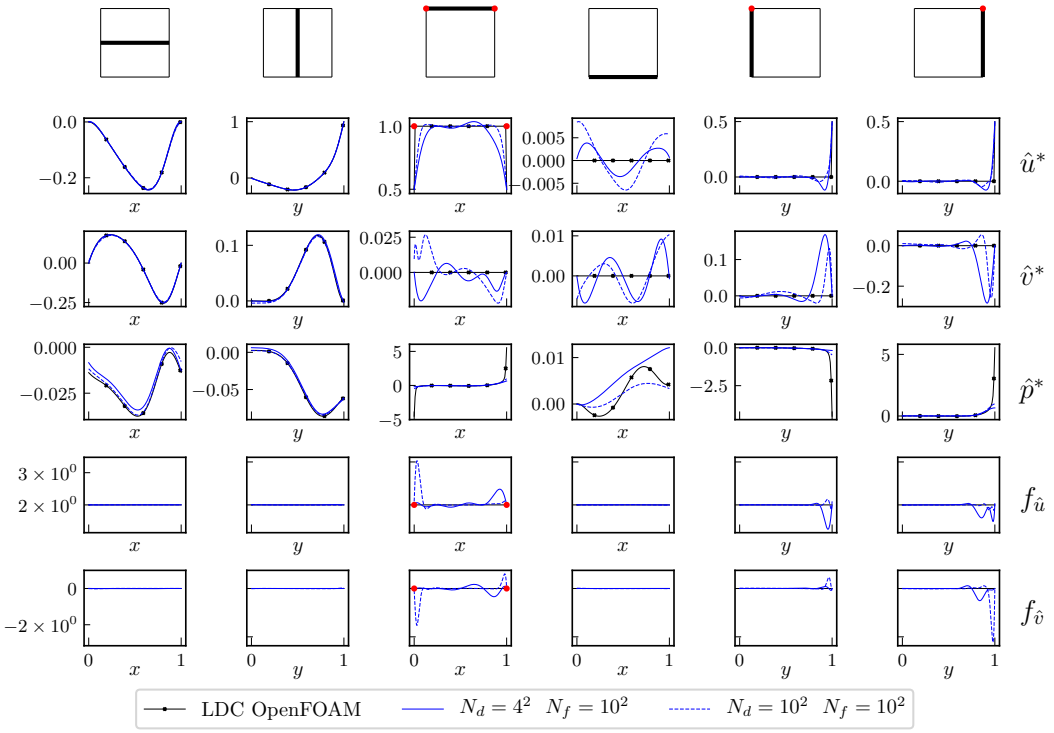


Figure 5.25: DPFINN divergence-free profiles for LDC case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 4^2$ and $N_d = 10^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE loss function used for both data and physics and Navier-Stokes equations in *NN*-form.

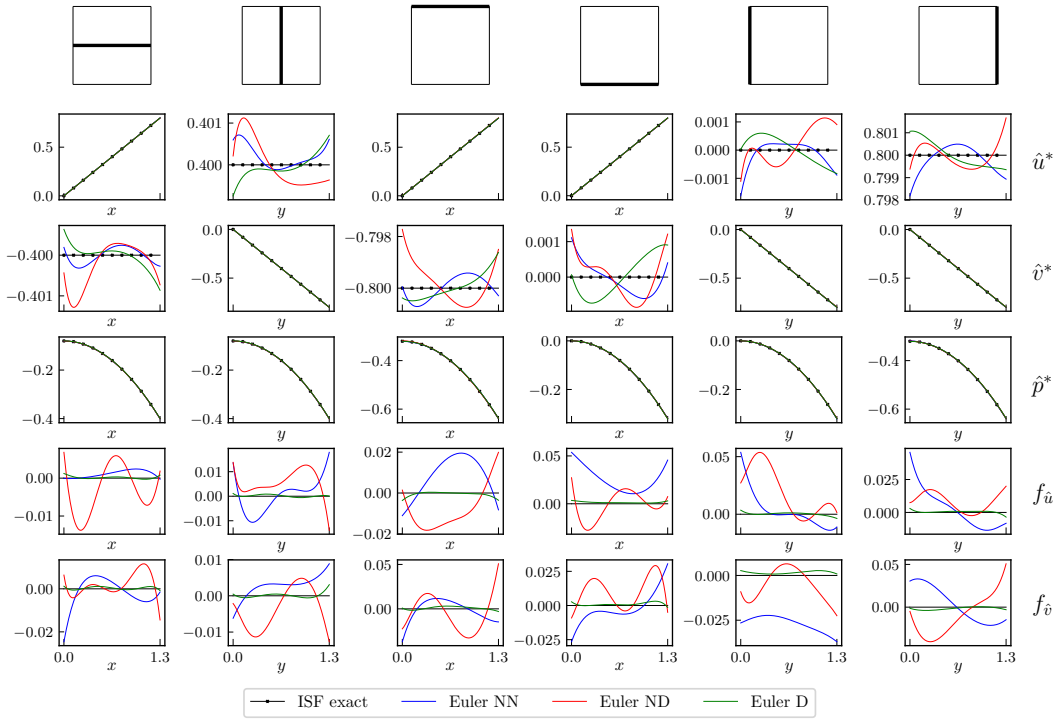


Figure 5.26: DPFINN divergence-free profiles for ISF case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE for data and for physics. Euler equations in *NN*-form, *ND*-form and *D*-form

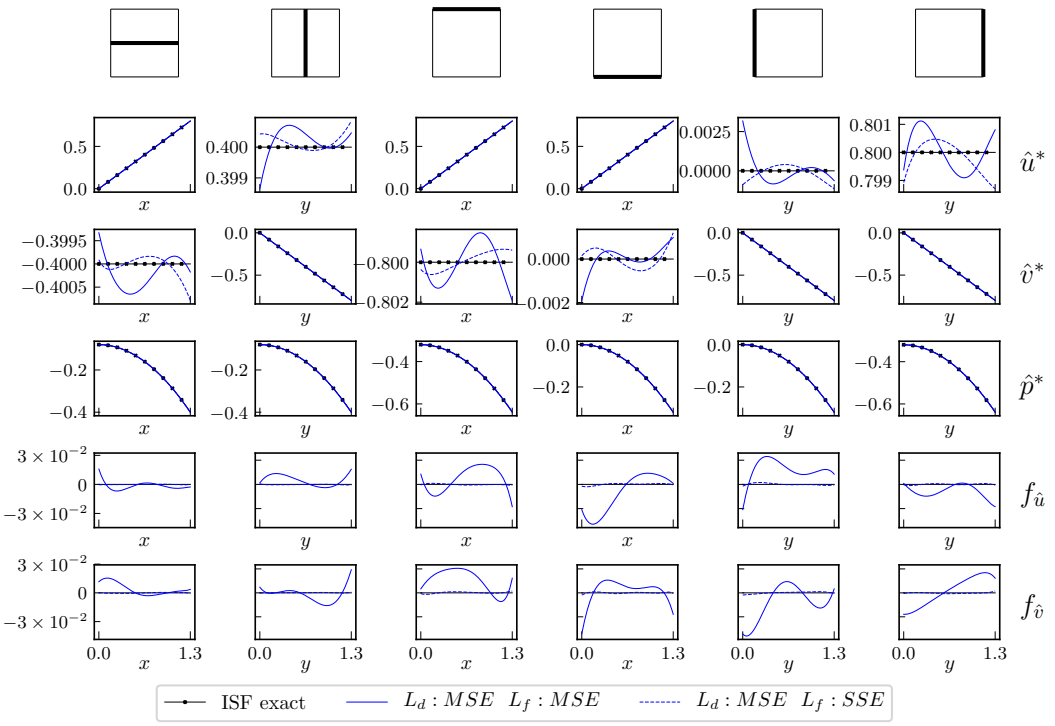


Figure 5.27: DPFINN divergence-free profiles for ISF case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE for data, MSE and SSE for physics. Euler equations in NN-form.

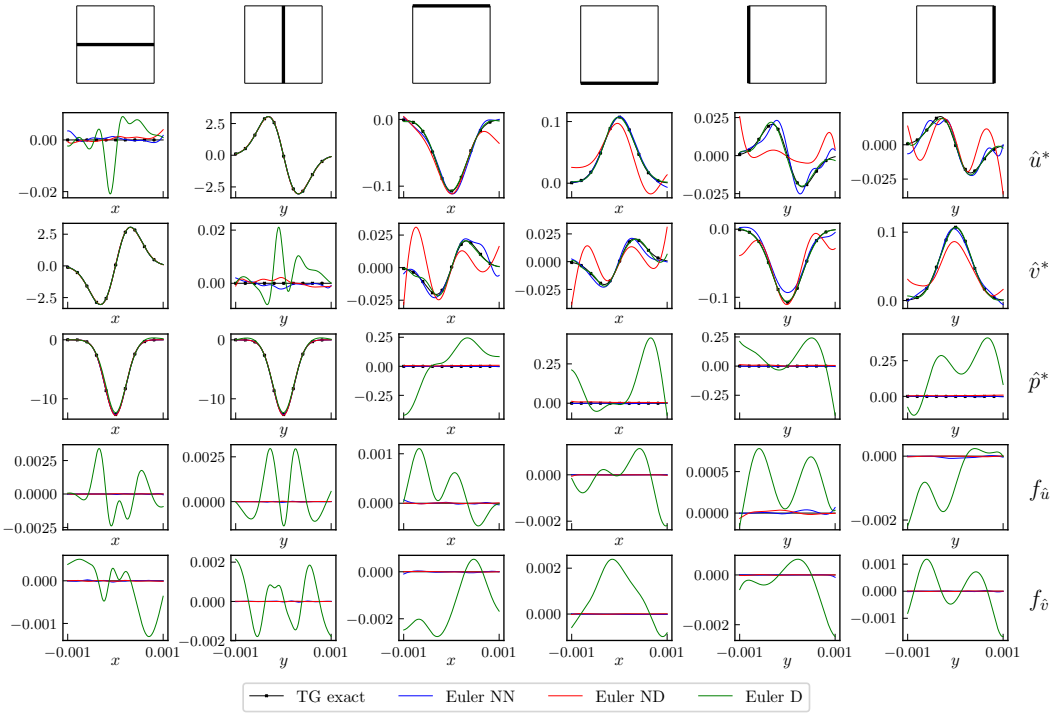


Figure 5.28: DPFINN divergence-free profiles for TG case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE for data and for physics. Euler equations in NN-form, ND-form and D-form

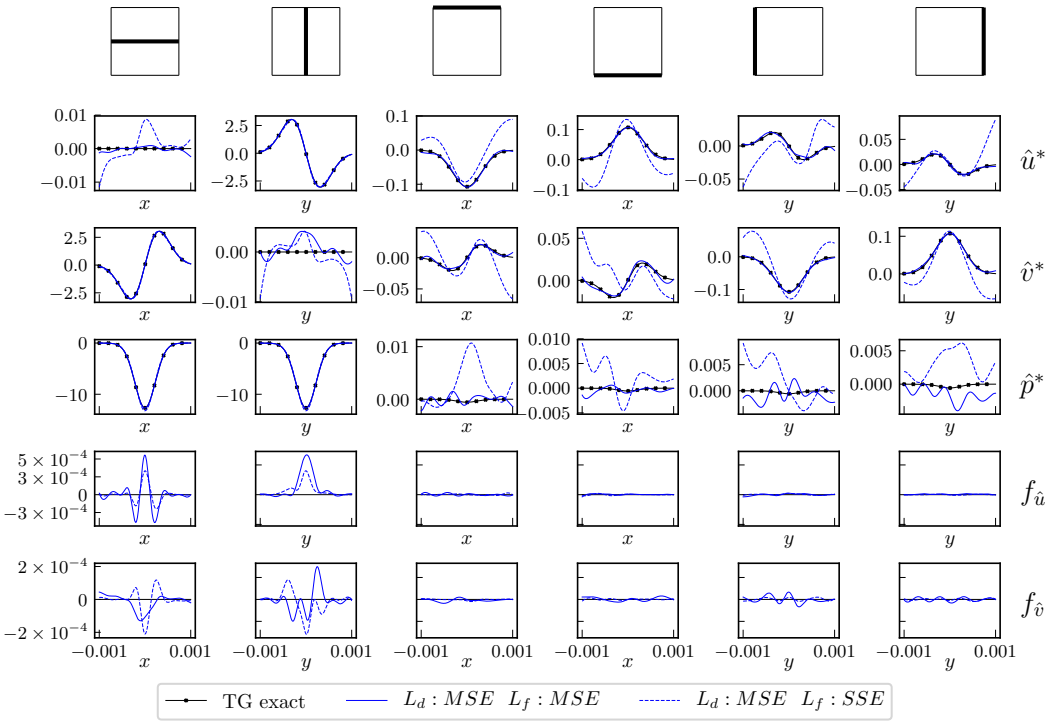


Figure 5.29: DPFINN divergence-free profiles for TG case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE for data, MSE and SSE for physics. Euler equations in *NN*-form.

By changing the loss function type of the physics loss function from the MSE to the SSE type, again for $N_d = 10^2$, the profiles can be examined, which are shown in Figure 5.29. The predictions of the SSE loss function type for the physics (L_f) produces inaccurate results for the boundaries, while they are good for the MSE physics inclusion. However, for the boundaries, the momentum equation discrepancies are reduced slightly and oscillate less. This shows that increasing the balance towards SSE reduces the importance of the data points on the boundaries too much. For the center profiles of all the flow variables, the differences are small.

5.4.3 Hiemenz Stagnation Flow

The results earlier show that the framework is able to infer the profiles of the flow variables well. However, the reduction of the momentum equation discrepancies for the Navier-Stokes equations included in the loss function in *NN*-form can be improved as can be concluded from the profiles in Figure 5.30. Here, the *ND*-form of the Navier-Stokes equations performs better, while the best results are obtained by the Navier-Stokes equations in *D*-form. This could be because the inverse scaling from the *NN*-form to the *D*-form balances out the total loss function better which also results in flow fields that are accurately predicted.

Earlier, the Hiemenz stagnation flow was tested with a very low amount of data points $N_d = 3^2$. The results were reasonable. The discrepancies were attempted to be minimized by increasing the amount of additional physics collocation points N_f, which did not improve the quality of the results, compared to the one with lower N_f. Another possibility is to put more emphasis on the physics loss function by altering the loss function. By choosing the SSE loss function for the physics inclusion instead of the MSE loss function, indeed the discrepancy of the momentum equations can be reduced, as shown by the profiles in Figure 5.31. Moreover, predictions of the $\hat{v}^*$ velocity are increased slightly. Therefore, this is a case where altering the loss function type proved to be beneficial for the prediction quality.

5.4.4 Lid Driven Cavity Flow

Varying the forms of the Navier-Stokes equations in the loss function of the physics is not expected to give results that vary significantly. This is because the reference values, as well as the scaling factors,

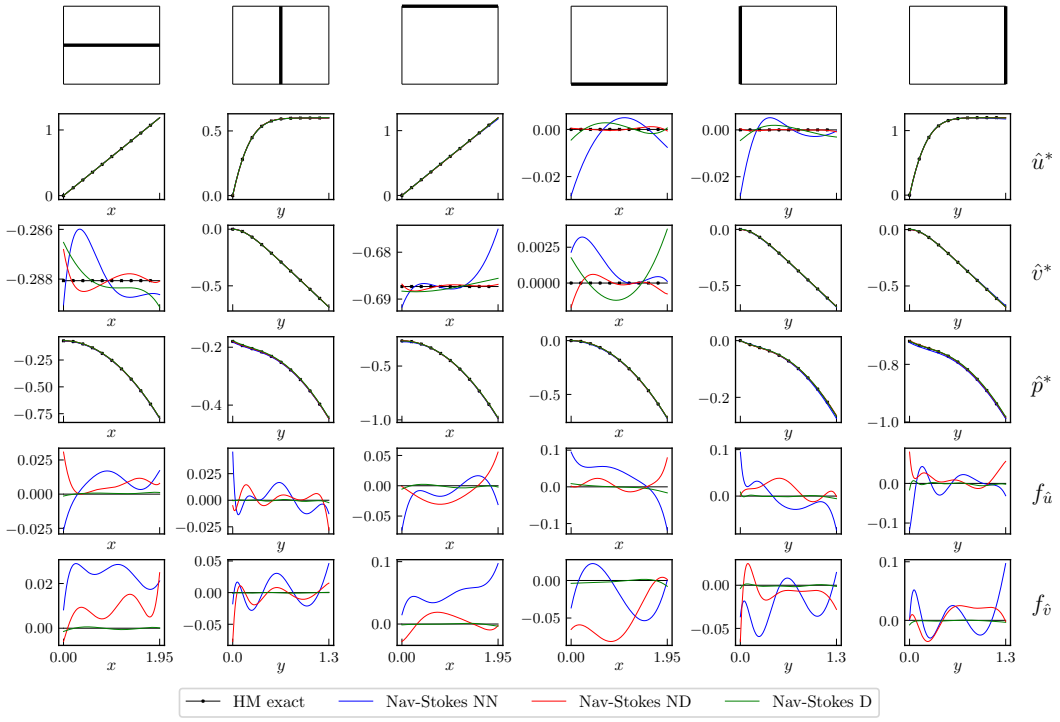


Figure 5.30: DPFINN divergence-free profiles for HM case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE for data and for physics. Navier-Stokes equations in *NN*-form, *ND*-form and *D*-form

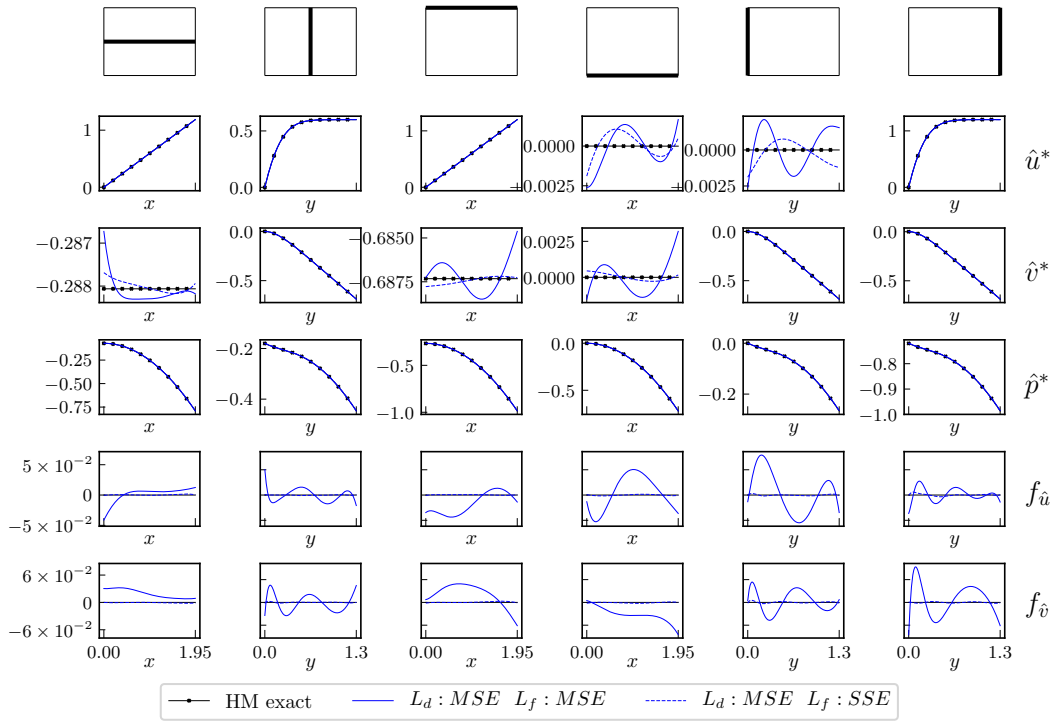


Figure 5.31: DPFINN divergence-free profiles for HM case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE for data, MSE and SSE for physics. Navier-Stokes equations in *NN*-form.

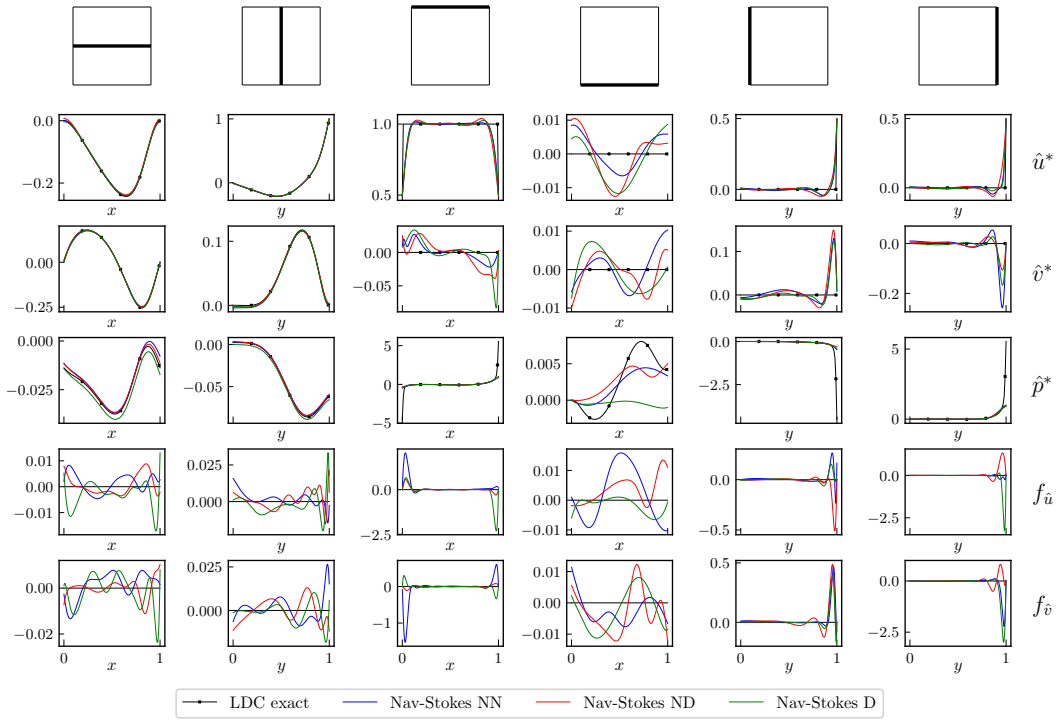


Figure 5.32: DPFINN divergence-free profiles for LDC case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 11^2$, amount of additional physics collocation points: $N_f = 100^2$. MSE for data and for physics. Navier-Stokes equations in *NN*-form, *ND*-form and *D*-form

are all close to 1 from the start. Nevertheless, the profiles are shown in Figure 5.32. What can be concluded is that the main differences are again to be found at the top corners. It seems that the *ND*-form of the Navier-Stokes equations produce momentum equation discrepancies along the lid, close to the corners that are lower than for the *NN*- and *D*-forms. However, throughout the domain the differences are negligible. As mentioned before, the discontinuities on the corners still give issues for the framework. This does not change with the different physics form inclusions.

The same is true for the different loss function formulations for the physics. Just as for the other cases, the usage of the SSE loss function for the physics, in comparison to using the MSE loss functions, the momentum equations discrepancies but slightly at the expense of flow field inference. This can be seen from Figure 5.33.

5.5 Effect of floating-point type

Returning to a comment that is already made, which is about the floating-point type (see Section 5.1). For all of the results in the previous section, the `tf.float64` floating-point type was used. The other option would be to use one that would give less precision: `tf.float32`. The L-BFGS-B optimizer is set to run until the differences between the iterations are smaller or equal to the resolution of the floating-point types. Therefore, the `tf.float64` floating-point type would increase the precision but also significantly increase the training time as the convergence criterion is much smaller. To compare the differences, the lid-driven cavity flow test case is taken, where an artificial neural network of $N_N = 10$, $La = 3$ is taken and $N_d = 10^2$ with additional physics collocation points $N_f = 10^2$, furthermore, the loss function types used are MSE and MSE for both physics and data and the Navier-Stokes equations in *NN*-form are used. As already mentioned, the maximum divergence error for the `tf.float32` floating-point type used will be approximately 10^{-7} and 10^{-16} for the `tf.float64` floating-point type used. However, the floating-point type does not only affect the maximum divergence error, but it also affects the other flow variables' inference performance. To illustrate this, the profiles for the lid-driven cavity flow test case with the different floating-point types are shown in Figure 5.34. Looking at the $\hat{u}^*$-velocity at the lid, the results from the `tf.float64` case are better than for `tf.float32`. The same applies to the other variables and the momentum equations near the top corners. This increased accuracy

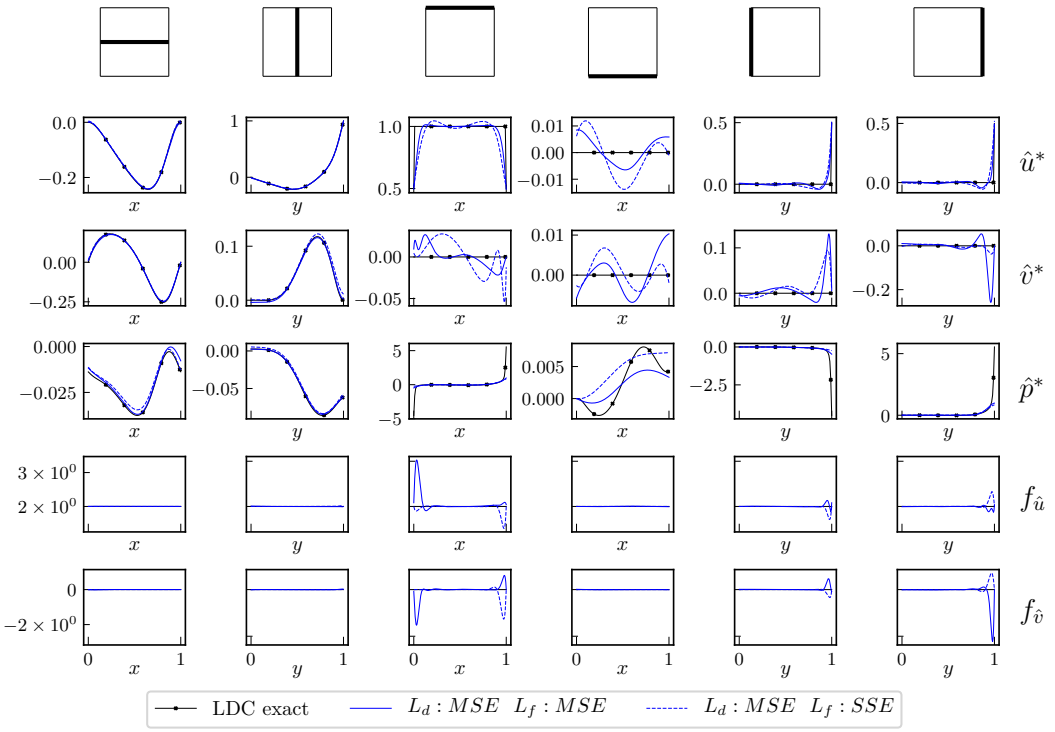


Figure 5.33: DPFINN divergence-free profiles for LDC case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 11^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE for data, MSE and SSE for physics. Navier-Stokes equations in *NN*-form.

comes at a considerate increase in training time: from about 30 minutes for `tf.float32` to 8 hours for the `tf.float64` case. This is most likely because of the additional gradients required because of the divergence-free basis.

These differences cannot only be found in the viscous cases, where the Navier-Stokes equations require an additional step of differentiation, but also for the inviscid cases. To illustrate this, the Taylor-Green vortex case is taken, with the same artificial neural network and sampling as for the lid-driven cavity case. The profiles are shown in Figure 5.35. Apart from the velocity and pressure profiles in the middle of the domain, large variations can be found on the boundaries, just as for the lid-driven cavity flow case. Moreover, again a similar increase in training time is found: from about 7 minutes for `tf.float32` to 3 hours for the `tf.float64` case.

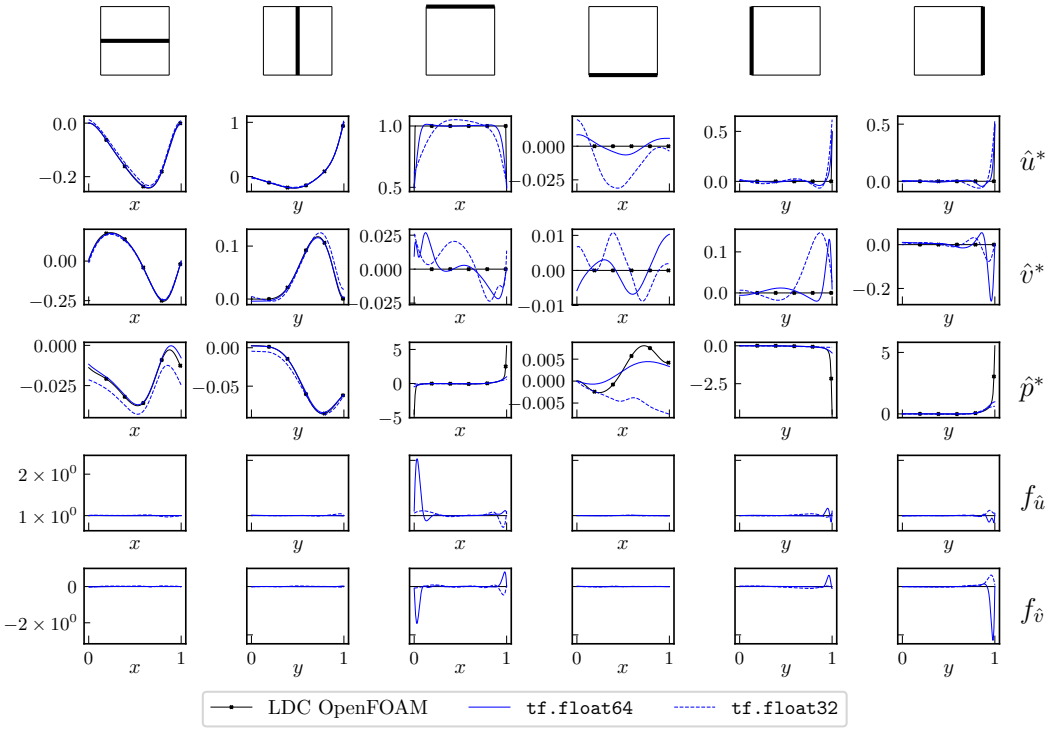


Figure 5.34: DPFINN divergence-free profiles for LDC case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 11^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE for data and for physics. Navier-Stokes equations in *NN*-form and different floating-point types:

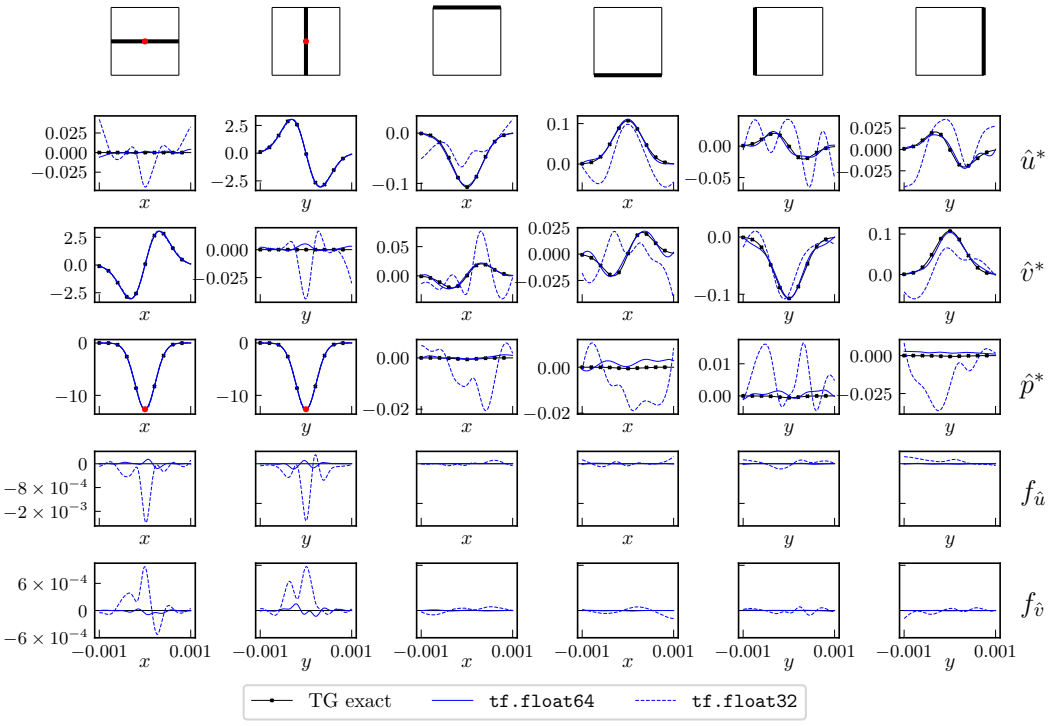


Figure 5.35: DPFINN divergence-free profiles for LDC case with artificial neural network: $N_n = 10$, $La = 3$. Data sampling is $N_d = 10^2$, amount of additional physics collocation points: $N_f = 10^2$. MSE for data and for physics. Euler equations in *NN*-form and different floating-point types:

6

Conclusions and Recommendations

6.1 Conclusions

To conclude this work, the conclusions are summarized in the next subsections. Where the DPFINN framework (subsection 6.1.1), as well as the test cases (subsection 6.1.2), are briefly summarized again. Furthermore, the conclusions of the results are mentioned in subsections 6.1.3 to 6.1.7. These conclusions along with the original research questions will be used to draw a final conclusion in subsection 6.1.8.

6.1.1 DPFINN framework

In this research, the data physics informed neural network (DPINN) framework was described with a special emphasis on fluid flow problems, which yielded the data physics fluids informed neural network (DPFINN) framework. This framework is based on the works of [37] and [37] and is tested on a variety of fluid flow problems. The fluid flow problems considered were all incompressible, therefore the incompressible Navier-Stokes equations were valid. Some cases were inviscid for which the simplified Navier-Stokes equations: the Euler equations were applied. However, both the Navier-Stokes equations and the Euler equations include the continuity equation which states that the divergence of the velocity field has to be equal to zero. In order to have the framework make predictions for the velocity fields that are divergence-free, a divergence free basis was employed and augmented to the DPFINN framework. For the training of the artificial neural networks, a combination of the Adam optimizer and the L-BFGS-B optimizer was used which proved to be a robust training strategy. Furthermore, the tanh activation function was used for the hidden artificial neurons. The framework was coded in `TensorFlow` and the test cases were run on GPU architectures.

6.1.2 Test cases

The fluid flow problems that are used to test the performance of a data physics fluid informed neural networks (DPFINN) are the inviscid stagnation flow, Taylor-Green vortex, Hiemenz Stagnation Flow and lid-driven cavity flow. All these test cases are 2D test cases and either have analytical solutions or have solutions obtained by performing CFD simulations (OpenFOAM). Moreover, the first two are inviscid and the latter two are viscous. These test cases are devised to assess the framework capabilities. It is noteworthy that, for the framework, the source of the velocity field is irrelevant, as long as the flow characteristics are known. Therefore, these test cases can be considered as test cases prior to applying the framework to more complicated flow fields e.g. experimental flow fields.

6.1.3 Divergence-free basis

The results for various flow test cases applied to the DPFINN framework were given. First, the divergence-free basis was tested. The DPFINN with the divergence-free basis predicted good results and predicted velocity fields that are divergence-free for all the test cases. The maximum divergence of the predicted velocity fields was corresponding to the resolution of the floating-point types used and therefore can be considered negligibly small. Furthermore, the discrepancies of the flow variables over the domain are similar to the test cases tested with the framework that does not feature the divergence-free basis. The same applies to the discrepancies in the momentum equations, they are similar for both

frameworks. Therefore, the addition of the divergence-free basis is not concluded to worsen the pressure inference capabilities. However, the divergence-free basis requires two additional steps of differentiation, which increases the training time of the cases significantly. The training times are roughly a factor of five to nine times higher. This difference is mainly because of the additional gradient steps, which yield extra graph complexity. This is especially true for the loss functions. When using a divergence-free basis, not only does the loss function for the physics contain gradients of the artificial neural networks, but so does the loss function for the data (as the velocity field is the result of the double curl of a potential). This increases the complexity of the total loss function significantly. Nevertheless, the results for the discrepancy of the flow variables as well as the momentum equations are of the same order for all the test cases and the framework with and without the divergence-free basis.

6.1.4 Architecture complexity

Next, the artificial neural networks that are suitable for the fluid variable inference proved to vary with each test case. The simplest test cases proved to be the stagnation flow cases. Which are the inviscid stagnation flow and the Hiemenz stagnation flow, the smaller artificial neural networks (with $N_n = 4$ and $La = 1$ or $La = 2$) already gave good results and this was not improved much by increasing the artificial neural network complexity. A case where a very small artificial neural network did not work, is the Taylor-Green vortex where at least a $N_n = 10$ and $La = 3$ was required. For the lid-driven cavity flow case, the results for the different artificial neural network did not vary much apart from the discrepancies near the corners of the lid. At these locations, the discontinuities of the horizontal velocity at the lid (u) yielded large discrepancies in the momentum equations as well as for the v-velocity. This was believed to be because of the divergence free basis, as the large local gradients for the velocities had to be canceled by each other for the velocity field to be divergence free.

6.1.5 Sampling

For all the test cases, a minimal amount of velocity data points is required. Replacing data points by collocation points or increasing the collocation points was showed to not necessarily improve the inference accuracy. Thus, they do not provide enough leverage to be able to increase the output accuracy in the gaps between the data points. Moreover, adding more collocation points and/or data often complicates the loss function significantly such that even though the minimization on the given velocity field data points is done successfully, the results outside of the training sets are often worse when more physics collocation points in the training sets were used. Nevertheless, for the Hiemenz viscous stagnation flow, a very low amount of data points already sufficiently informed the framework and the flow fields predicted by the artificial neural network matched the exact solution well. However, not for the wall, where the predicted velocity fields showed a slight slip, where the no-slip condition applies. For the lid-driven cavity flow, the corners again seemed to be troublesome and the addition of data points did help in terms of the flow field inference, just not for the minimization of the physics near the corners. Throughout this work, most of the problems are assumed to be well-posed because the velocity field data provided is deemed sufficient. This is also the reason why additional Dirichlet boundary conditions were not considered, as it merely adds data points.

6.1.6 Loss function

As increasing the physics collocation points showed to not significantly improve the accuracy of the predictions, the balancing of the loss function between the data and the physics was analyzed. More specifically, the alteration of the physics loss function, as well as the different physics forms, were considered. By varying both, it became clear that minimizing both the data and the velocity field is difficult. This seemed to be the case for all test cases. For example, by taking a SSE loss function for physics while keeping the loss function for the data fixed at MSE, minimization of the physics is more favored (depending on the sizes of N_d and N_f). This often resulted in worsened velocity field inference. Thus, if the physics is favored in the loss function, the framework often tries to converge to a solution that results in flow fields that are constant, which will satisfy the governing equations as the trivial solution. It appears that the data part of the loss function is in those cases not able to counteract this. Perhaps the general assumption of $\hat{\mathbf{q}} \approx \mathbf{q}$ is valid but the same can maybe not be said about the gradients of both and therefore also not for equation (3.4).

6.1.7 Physics forms

Next to the scaling of the loss functions, the different physics implementations prove to affect the performance of the framework as well. However, generally, the physics form that uses the unit range form (NN) provided the best trade-off compared to the non-dimensional (ND) and the dimensional forms (D), albeit not for all cases. The reason for this is most likely because the NN physics form is in the same dimensional space as the one used for the data loss function and therefore balances the loss function in a better way.

6.1.8 Final conclusion

Nevertheless, the divergence free basis ensures that the velocity fields that are obtained by the framework are divergence free. However, the higher training time most likely does not outweigh the added benefit of the velocity fields being divergence free. Especially for more complicated cases e.g. experimental cases, the added training time will most likely not make this approach worthwhile. Especially if the experimental velocity field contains significant amounts of data points. On top of that, the floating-point type seemed to have an effect on the results on the boundaries of the flow case domains in the sense that higher floating-point types gave better results but also significantly increased training times even further. Next, the research questions from Section 1.3 are revisited below where the answers are distilled from the conclusions:

Main Research Question 1: *How can artificial neural networks be used to infer flow fields from given velocity field data and flow physics?*

- **Research question 1:** *Which artificial neural networks are suitable for fluid flow data assimilation purposes?*
  This depends greatly on the fluid flow application. In general, the number of features e.g. vortices or boundary layer phenomena from the velocity field data could give an indication for the complexity of the artificial neural network that is required. The reason for this is that the artificial neural networks constructs regions by subdividing the spatial domain. The more features a flow contains, the more complexity is required.

- **Research question 2:** *Which loss function choices are the best for the implementation of the data and physics?*
  The types that exist are e.g. MSE and SSE. Where the different combinations give different scalings to the loss function terms and thus balances the loss function differently. However, for some cases, the usage of MSE for the data and SSE for the physics loss function proved to be the best.

- **Research question 3:** *Which machine learning algorithms are appropriate for learning artificial neural networks for flow field inference?*
  A combination of a stochastic optimizer (Adam) and a gradient-descent (L-BFGS-B) optimizer algorithm was used throughout this work and proved to be robust and produce consistent results.

Main Research Question 2: *How can the Navier-Stokes equations including mass conservation be implemented in the framework?*

- **Sub-question 1:** *What is the best formulation of the Navier-Stokes equations to be used in the loss function?*
  For the physics, the governing equations can be scaled accordingly to scale the physics part of the loss function. The NN-form proved to be the one that scales the governing equations such that it provided the best balance between physics and data inference for most of the cases.

- **Sub-question 2:** *How can the requirement of the divergence-free velocity fields be satisfied by the framework?*
  This can be done by the divergence-free basis. Which reduced the maximum absolute divergence of any test case down to the floating-point type resolution, while not significantly worsening the inference performance in case no divergence-free basis was used.

As an overall conclusion to this work, the following can be said: The divergence-free basis produced good results and divergence-free velocity fields. However, the penalty for the utilization of the basis is increased training times. However, the framework produced good results for all the test cases. Furthermore, there is not one physics form that produces the best results, however, the *NN*-form often gave the best trade-off between physics and data. This can be improved for some cases by using a SSE loss function type for the physics loss function, instead of the MSE loss function. With regard to the architecture complexity, the best-recommended practice is to initially test with simple(r) artificial neural networks and base the choice of artificial neural network complexity on the expected amount of flow features.

6.2 Recommendations for Future Work

Because of the difficulty of minimizing the fluid flow equation discrepancy and simultaneously the test data, it is therefore recommended to keep investigating this approach with the emphasis on minimizing the fluid flow equations while maintaining good data discrepancy minimization. An option to investigate is to use tailor-made activation functions (instead of the tanh or other common ones), which are able to generate momentum equations that minimize well with the given data in the loss function and where the balance of the loss functions becomes less of a stringent problem. The way new activation functions can be obtained is to look closely at the equations the artificial neural network can generate and how the derivatives are formed and vary with different activation functions. This becomes difficult as the artificial neural networks are hard to understand, nevertheless some attempts to it are made in this work and can be continued. Nevertheless, the key focus should be the relation between the different activation functions and the physics inclusion. One of the possibilities that can be researched further is suggested by [37], here the sinusoidal activation functions were used. This could be of interest because the higher order derivatives of the sinusoidal functions either result in cosinusoidal or sinusoidal functions. However, the performance of these within the hidden layers of the artificial neural networks is unclear, especially in conjunction with the backpropagation algorithm. Another option that is not encountered and not researched here is to optimize and distribute different types of activation functions throughout the artificial neural networks, but again the performance largely depends on the backpropagation algorithm applicability.

Nevertheless, the approximation capabilities of the artificial neural networks are large and should remain a topic for research, also in the fluid flow data assimilation context. However, understanding of the artificial neural networks remains paramount to the progress. Moreover, to test the flow field prediction quality by the artificial neural networks, not only the discrepancy of the flow fields should be examined, but also the discrepancy of the fluid flow equations. As the discrepancies of both provide the best analysis of the results, in case additional uncertainty quantification estimates are absent.

To conclude this work with a more general comment: The machine learning field has grown substantially and the interest of the machine learning field to physics problems is not surprising. This is mainly due to the fact that the models used in the machine learning fields are powerful and the approximation capabilities are big. This comes at the price of a reduced understanding of how solutions are formed. Therefore, using the models for physics problems requires additional efforts on quantifying the error made in the physical governing equations. Once a better understanding of the machine learning models (not just artificial neural networks) and the physics in the context of those models are obtained, this approach can be considered a valuable alternative.

Bibliography

[1] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] John David Anderson and J Wendt. *Computational fluid dynamics*, volume 206. Springer, 1995.

[3] John David Anderson Jr. *Fundamentals of aerodynamics*. Tata McGraw-Hill Education, 2010.

[4] Iliass Azijli and Richard P. Dwight. Solenoidal filtering of volumetric velocity measurements using Gaussian process regression. *Experiments in Fluids*, 56(11):1–18, 2015. ISSN 07234864. doi: 10.1007/s00348-015-2067-7.

[5] I.A Basheer and M Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, dec 2000. ISSN 01677012. doi: 10.1016/S0167-7012(00)00201-3. URL `http://linkinghub.elsevier.com/retrieve/pii/S0167701200002013`.

[6] Atilim Gunes Baydin, Barak A. Pearlmutter, Alexey Andreyevich Radul, and Jeffrey Mark Siskind. Automatic differentiation in machine learning: a survey. 2015. ISSN 15337928. URL `http://arxiv.org/abs/1502.05767`.

[7] M Baymani, S Effati, H Niazmand, and A Kerayechian. Artificial neural network method for solving the Navier – Stokes equations. *Neural Computing and Applications*, pages 765–773, 2015. ISSN 0941-0643. doi: 10.1007/s00521-014-1762-2. URL `http://dx.doi.org/10.1007/s00521-014-1762-2`.

[8] Modjtaba Baymani, Asghar Kerayechian, and Sohrab Effati. Artificial Neural Networks Approach for Solving Stokes Problem. *Applied Mathematics*, 01(04):288–292, 2010. ISSN 2152-7385. doi: 10.4236/am.2010.14037. URL `http://www.scirp.org/journal/doi.aspx?DOI=10.4236/am.2010.14037`.

[9] Jens Berg and Kaj Nyström. A unified deep artificial neural network approach to partial differential equations in complex geometries. *Neurocomputing*, 317:28–41, 2018. ISSN 18728286. doi: 10.1016/j.neucom.2018.06.056. URL `https://doi.org/10.1016/j.neucom.2018.06.056`.

[10] Tim Dockhorn. A Discussion on Solving Partial Differential Equations using Neural Networks. 2019. URL `http://arxiv.org/abs/1904.07200`.

[11] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, July 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2021068`.

[12] Roger Fletcher. *Practical methods of optimization*. John Wiley & Sons, 2013.

[13] N Fujisawa, S Tanahashi, and K Srinivas. Evaluation of pressure field and fluid forces on a circular cylinder with and without rotational oscillation using velocity data from piv measurement. *Measurement Science and Technology*, 16(4):989, 2005.

[14] UKNG Ghia, Kirti N Ghia, and CT Shin. High-re solutions for incompressible flow using the navier-stokes equations and a multigrid method. *Journal of computational physics*, 48(3):387–411, 1982.

[15] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. 9:249–256, 2010.

[16] W Gropp, A Khodadoust, J Slotnick, D Mavriplis, D Darmofal, J Alonso, and E Lurie. CFD Vision 2030 Study: A Path to Revolutionary Computational Aerosciences. Technical report, 2014.

[17] S. Haykin. *Neural Networks and Learning Machines*. 2008. ISBN 9780131471399. doi: 978-0131471399.

[18] Karl Hiemenz. Die grenzschicht an einem in den gleichformigen flussigkeitsstrom eingetauchten geraden kreiszylinder. *Dinglers Polytech. J.*, 326:321–324, 1911.

[19] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 08936080. doi: 10.1016/0893-6080(89)90020-8.

[20] Vugar E. Ismailov. On the approximation by neural networks with bounded number of neurons in hidden layers. *Journal of Mathematical Analysis and Applications*, 417(2):963–969, 2014. ISSN 10960813. doi: 10.1016/j.jmaa.2014.03.092. URL `http://dx.doi.org/10.1016/j.jmaa.2014.03.092`.

[21] A.K. Jain, Jianchang Mao, and K.M. Mohiuddin. Artificial neural networks: a tutorial. *Computer*, 29(3):31–44, mar 1996. ISSN 00189162. doi: 10.1109/2.485891. URL `http://ieeexplore.ieee.org/document/485891/`.

[22] Bo-nan Jiang. A least-squares finite element method for incompressible navier-stokes problems. 14 (September 1989):843–859, 1992.

[23] Bo-nan Jiang. *The least-squares finite element method: theory and applications in computational fluid dynamics and electromagnetics*. Springer Science & Business Media, 1998.

[24] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. pages 1–15, 2014. ISSN 09252312. doi: http://doi.acm.org.ezproxy.lib.ucf.edu/10.1145/1830483.1830503. URL `http://arxiv.org/abs/1412.6980`.

[25] A. K. Kolmogorov. On the representation of continuous functions of several variables by superposition of continuous functions of one variable and addition. *Doklady Akademii Nauk SSSR*, 114: 369–373, 1957.

[26] Vladik Ya Kreinovich. Arbitrary nonlinearity is sufficient to represent all functions by neural networks: A theorem. *Neural Networks*, 4(3):381–383, 1991. ISSN 08936080. doi: 10.1016/0893-6080(91)90074-F.

[27] Manoj Kumar and Neha Yadav. Multilayer perceptrons and radial basis function neural network methods for the solution of differential equations: A survey. *Computers and Mathematics with Applications*, 62(10):3796–3811, 2011. ISSN 08981221. doi: 10.1016/j.camwa.2011.09.028. URL `http://dx.doi.org/10.1016/j.camwa.2011.09.028`.

[28] I.E. Lagaris, A. Likas, and D.I. Fotiadis. Artificial neural networks for solving ordinary and partial differential equations. *IEEE Transactions on Neural Networks*, 9(5):987–1000, 1998. ISSN 10459227. doi: 10.1109/72.712178. URL `http://ieeexplore.ieee.org/document/712178/`.

[29] Isaac Elias Lagaris, Aristidis C. Likas, and Dimitrios G. Papageorgiou. Neural-network methods for boundary value problems with irregular boundaries. *IEEE Transactions on Neural Networks*, 11(5):1041–1049, 2000. ISSN 10459227. doi: 10.1109/72.870037.

[30] Randall J LeVeque et al. *Finite volume methods for hyperbolic problems*, volume 31. Cambridge university press, 2002.

[31] Warren S. McCulloch and Walter H. Pitts. originally published in: Bulletin of Mathematical Biophysics, Vol. 5, 1943, p. 115-133. *Bulletin of Mathematical Biophysics*, 5:115–133, 1943. ISSN 0007-4985. doi: 10.1016/j.joca.2006.06.009.

[32] Francis J. Narcowich and Joseph D. Ward. Generalized Hermite Interpolation Via Matrix-Valued Conditionally Positive Definite Functions. *Mathematics of Computation*, 63(208):661, 2006. ISSN 00255718. doi: 10.2307/2153288. URL http://www.ams.org/jourcgi/jour-getitem?pii=S0025-5718-1994-1254147-6.

[33] Jorge Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35(151):773–773, sep 1980. ISSN 0025-5718. doi: 10.1090/S0025-5718-1980-0572855-7. URL http://www.ams.org/jourcgi/jour-getitem?pii=S0025-5718-1980-0572855-7.

[34] Jorge Nocedal and Stephen Wright. *Numerical optimization*. Springer Science & Business Media, 2006.

[35] Ivan Nunes da Silva, Danilo Hernane Spatti, Rogerio Andrade Flauzino, Luisa Helena Bartocci Liboni, and Silas Franco dos Reis Alves. *Artificial Neural Networks*. 2017. ISBN 978-3-319-43161-1. doi: 10.1007/978-3-319-43162-8. URL https://link.springer.com/content/pdf/10.1007{%}2F978-3-319-43162-8.pdf{%}0Ahttp://link.springer.com/10.1007/978-3-319-43162-8.

[36] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations. (Part I):1–22, 2017. URL http://arxiv.org/abs/1711.10561.

[37] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. Hidden Fluid Mechanics: A Navier-Stokes Informed Deep Learning Framework for Assimilating Flow Visualization Data. 2018. doi: arXiv:1808.04327v1. URL http://arxiv.org/abs/1808.04327.

[38] Russell Reed and Robert J MarksII. *Neural smithing: supervised learning in feedforward artificial neural networks*. Mit Press, 1999.

[39] Hermann Schlichting et al. *Boundary layer theory*, volume 960. Springer, 1960.

[40] Jan F G Schneiders, Richard P Dwight, and Fulvio Scarano. Time-supersampling of 3D-PIV measurements with vortex-in-cell simulation. *Experiments in Fluids*, 55(3), 2014. ISSN 07234864. doi: 10.1007/s00348-014-1692-x.

[41] Martin Schwabe. Über druckermittlung in der nichtstationären ebenen strömung. *Ingenieur-Archiv*, 6(1):34–50, 1935.

[42] Alexander Schwarz and Richard P Dwight. Data assimilation for navier-stokes using the least-squares finite-element method. *International Journal for Uncertainty Quantification*, 8(5), 2018.

[43] R. Shekari Beidokhti and A. Malek. Solving initial-boundary value problems for systems of partial differential equations using neural networks and optimization techniques. *Journal of the Franklin Institute*, 346(9):898–913, 2009. ISSN 00160032. doi: 10.1016/j.jfranklin.2009.05.003. URL http://dx.doi.org/10.1016/j.jfranklin.2009.05.003.

[44] Brendan D. Tracey, Karthikeyan Duraisamy, and Juan J. Alonso. A Machine Learning Strategy to Assist Turbulence Model Development. *53rd AIAA Aerospace Sciences Meeting*, (January), 2015. ISSN 978-1-62410-343-8. doi: 10.2514/6.2015-1287. URL http://arc.aiaa.org/doi/10.2514/6.2015-1287.

[45] P van der Smagt. Minimisation methods for training feedforward neural networks. *Neural Networks*, 7(1):1–11, 1994. URL http://www.sciencedirect.com/science/article/pii/0893608094900523.

[46] P. L. van Gent, D. Michaelis, B. W. van Oudheusden, P. Weiss, R. de Kat, A. Laskari, Y. J. Jeon, L. David, D. Schanz, F. Huhn, S. Gesemann, M. Novara, C. McPhaden, N. J. Neeteson, D. E. Rival, J. F.G. Schneiders, and F. F.J. Schrijer. Comparative assessment of pressure field reconstructions from particle image velocimetry measurements and Lagrangian particle tracking. *Experiments in Fluids*, 58(4):1–23, 2017. ISSN 07234864. doi: 10.1007/s00348-017-2324-z.

[47] B. W. Van Oudheusden. PIV-based pressure measurement. *Measurement Science and Technology*, 24(3), 2013. ISSN 13616501. doi: 10.1088/0957-0233/24/3/032001.

[48] Bas W Van Oudheusden, Fulvio Scarano, Eric WM Roosenboom, Eric WF Casimiri, and Louis J Souverein. Evaluation of integral forces and pressure fields from planar velocimetry data for incompressible and compressible flows. *Experiments in Fluids*, 43(2-3):153–162, 2007.

[49] Holger Wendland. Divergence-Free Kernel Methods for Approximating the Stokes Problem. *SIAM Journal on Numerical Analysis*, 47(4):3158–3179, 2009. ISSN 0036-1429. doi: 10.1137/080730299.

[50] F. M. White. *Viscous fluid flow*. 2006. ISBN 0203888081. doi: 10.1007/s007690000247.