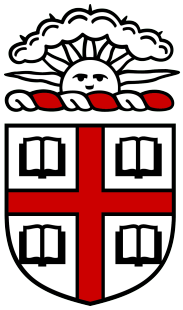
Boosting the Training of Physics-Informed Neural Networks with Transfer Learning

Marius Merkle

Bachelor Thesis towards the degree

B.Sc. Engineering Science



BROWN

Brown University

Department of Applied Mathematics

Providence, USA

Reviewer, Brown University:	George Em Karniadakis
Reviewer, Technical University of Munich:	Wolfgang Polifke
Supervisor:	Xuhui Meng

03/15/2020

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

Marius Merkle

03/15/2021

Date

Abstract

Differential equations are ubiquitous in science and engineering and their solutions are crucial for many applications as they can help us to understand various scientific disciplines described by applied mathematics. Unfortunately, differential equations are notoriously hard to solve. For almost all practically relevant differential equations, exact analytical solutions are unknown. Therefore, numerical solvers that yield solutions with accuracy up to a certain degree have been developed for several decades. Despite these tremendous efforts, high-performance computing clusters may still need prohibitively long (weeks to months) to carry out sophisticated numerical simulations. Therefore, the design of new efficient algorithms for differential equations is of primary importance and could have revolutionary effects in both academia and industry.

With the rise of artificial intelligence, artificial neural networks have emerged as one of the most promising data-driven algorithms. Recently, they have been applied to differential equations: *physics-informed neural networks* have been proposed as a new framework to solve differential equations numerically with artificial neural networks, but computational effort is still a bottleneck.

This bachelor's thesis outlines a new strategy to obtain numerical results with physics-informed neural networks in fraction of time, while pushing the accuracy to new levels. This boost in time and accuracy is possible through the application of transfer learning: instead of starting from a random initial guess, previously acquired knowledge on similar problems is transferred to a new related task. Empirical results demonstrate that this transfer of knowledge in physics-informed neural networks is beneficial for any two similarly related problems. On top, achieved boosts correlate with the underlying similarity of two problems, resulting in efficiency improvements of up to almost two orders of magnitude while outperforming other approaches in accuracy.

On a larger scale, these findings motivate the establishment of an open-source database of physics-informed neural networks. It will be shown that such a database would be self-reinforcing, i.e. their effectiveness would increase with the number of its entries. Therefore, large databases could potentially pave the road to boost physics-informed neural networks in several applications.

Contents

1 Introduction

As part of the contemporary resurgence of artificial intelligence (AI), data-driven algorithms have had revolutionary impact in several scientific disciplines, e.g. image classification [1], speech recognition [2] and natural language processing [3]. Yet, its most known disciplines such as machine learning (ML) and deep learning (DL) have only been adopted recently in the scientific computing community. This new field of scientific machine learning (SciML) is rapidly emerging and addresses the intersection of data-driven algorithms (e.g. DL, ML) and science and engineering applications [4].

Most notably, deep learning methods that are based on artificial neural networks have emerged as one of the most promising data-driven algorithms. They have been applied to virtually anything, more often than not with transformative impact, and recently to differential equations: *physics-informed neural networks* (PINNs) have been proposed as a framework to solve differential equations numerically with artificial neural networks. [5, 6, 7]: in strong contrast to conventional numerical approaches, the discussed physics-informed neural networks reformulate the task of solving a differential equation as an optimization problem.

PINNs have a variety of features that make them attractive for various applications in applied mathematics: First, they are so-called mesh-free algorithms. In conventional numerical approaches, including finite elements (FE), finite differences (FD) and finite volumes (FV), a discretization of the considered spatial and/or temporal domain, called meshing, is required. As most problems in practice are based on complex geometries, meshing is often an art by itself as the discrete volumes (cells) in a mesh need to be of regular shape. Hence, in computational science and engineering (CSE), this pre-processing step is often costly. PINNs also require discrete points on which the optimization problem should be solved, but the shapes of resulting volumes between points are of no particular interest. This results from the nature of the numerical algorithm as no fluxes across surfaces are calculated to enforce discrete conservation, e.g. as for FV [8].

Second, the solution of a PINN is available in differentiable and closed analytic form [9]. Conventional numerical approaches yield either solutions of limited differentiability or discrete instead of continuous solutions. A PINN can generalize well in between discrete data points and its derivatives may be used for subsequent steps in post-processing, e.g. the calculation of shear forces on abdominal aortic aneurysms which require the first derivative of a neural network output [10].

Third, PINNs provide a flexible framework that is applicable to many types of problems that arise in science and engineering. As such, it is possible to apply them to ordinary and partial equations as well as to systems constituted of them [9]. Interest-

ingly, from an implementation-wise perspective, forward and inverse problems only differ in minor changes in the code [8]. This is remarkable as previous approaches on inverse problems require a set of forward simulations to interpolate in between, presenting a solution strategy different from forward problems. In all, PINNs have emerged as a powerful technique in physics-informed deep learning with potentially revolutionary impact.

Its introduction in 2019 has sparked a number of researchers across the globe to apply PINNs to specific engineering fields including fluid flow [11, 12], cardiovascular flow [10], solid mechanics [13] and fractures [14]. In addition, many works have targeted computational efficiency and introduced different techniques to reduce effort and time via adaptive activation functions [15, 16] and heterogeneously weighted loss-functions to balance multi-objective optimization [17, 18].

Yet, one feature is inherent to all these works: optimization starts from *random initialization*. More precisely, in the widely adopted Xavier/Glorot optimizer, the initial weights and biases of a neural network are drawn from a random distribution [19]. The underlying uniform random distribution ensures that gradients that are computed during optimization are balanced across all layers and lead to effective network training. In turn, the initial prediction of a neural network is random, therefore not related to the problem and, in general, a bad guess. Stated differently, the intent of the Xavier/Glorot optimizer is to enable smooth gradient flow at the cost of an educated initial guess.

1.1 Idea Sketch

This bachelor's thesis alters the above-quoted paradigm by introducing a framework that starts from an *educated initial guess*, hence non-random initialization. The motivation behind this approach is the following: during training, neural networks *iteratively* improve the solution, i.e. the initial guess is continuously adapted towards the final guess (visualized in section 4.1). Therefore, a good initial guess as close as possible to the exact solution, should lead to faster optimization as long as the gradient flow dynamics remain smooth. This leads to the following hypothesis:

Hypothesis. *A neural network shows the trend that its predictions improve over time, i.e. approximate the exact solution better and better as training progresses. The closer the initial guess to the exact solution, the higher the speed of convergence, and possibly, the more accurate the final guess.*

The question then becomes how to come up with a good initial guess. The answer lies, at least on a macroscopic level, in the deterministic relationship between problem statement and physical solution. Given some problem in science and engineering, including geometry, constraints and material parameters, the resulting physical process (e.g.

fluid flow, solid deformation, temperature distribution) is a natural response. In almost all cases, the solution distribution may be seen as a continuous function of the problem setup. Small variations in the problem setup lead to small variations in the response. Therefore, if we know a priori that similar problem setups result in similar responses, it is natural to choose the final guess of some problem as an initial guess for a similar, related problem.

The final line of thoughts along with the above hypothesis forms the basis of this bachelor's thesis which is organized as follows: in section 2, the reader will be introduced to fundamental background knowledge with an overview on differential equations (section 2.1) and neural networks (section 2.2) that are combined and synthesized in physics-informed neural networks (section 2.3). Then, a framework grounded on the above argumentation is explained in detail (section 3) and subsequently numerically tested on a variety of examples (section 4). They will start with an intuitive, visual and vivid introduction (section 4.1), lead on to some benchmarking cases (sections 4.2, 4.3), and finish with a more sophisticated application (section 4.4).

2 Background

2.1 Differential Equations

2.1.1 Introduction

Differential equations are ubiquitous in science and engineering. They arise in fluid mechanics, thermodynamics, celestial motion, economics, biology and many more. The reason for their widespread impact is due to the nature of a differential equation: it relates a physical quantity of interest to its temporal and/or spatial rate of change, which is inherent to various natural phenomena [20].

Let us formalize this concept by introducing two independent variables: the three-dimensional spatial vector $\begin{pmatrix} x & y & z \end{pmatrix} \in \mathbb{R}^3$ and the scalar time $t \in \mathbb{R}$. The physical quantity of interest, e.g. $\mathbf{u}(x, y, z, t)$, is called a dependent variable as it is a function of space and time.

Differential Equation Any mathematical equation that involves at least one dependent variable and its derivatives with respect to at least one independent variable and whose solution is the dependent variable itself is called a differential equation:

$$\mathbf{u}_t + \mathcal{N}_\mathcal{P}(\mathbf{u}, x, y, z, t) = \mathcal{F}_\mathcal{P}(x, y, z, t) \tag{2.1}$$

$\mathcal{N}$ and $\mathcal{F}$ are non-linear operators parametrized by $\mathcal{P}$ and $\mathbf{u}_t = \frac{\partial \mathbf{u}}{\partial t}$ denotes the element-wise derivative of $\mathbf{u}$ with respect to time. In a similar fashion, $\mathbf{u}_x = \frac{\partial \mathbf{u}}{\partial x}$ and $\mathbf{u}_{xx} = \frac{\partial^2 \mathbf{u}}{\partial x^2}$ denote the first and second derivative with respect to x, respectively.

2.1.2 Classification

Differential equations can be described and classified by several characteristics which are listed in the following:

1. *ordinary* (one single dependent variable) vs. *partial* (more than one dependent variable)
2. *linear* ($\mathcal{N}$ is linear in $\mathbf{u}$ and its derivatives) vs. *non-linear* ($\mathcal{N}$ is non-linear in $\mathbf{u}$ and its derivatives)
3. *order* of a differential equation: the order of the highest derivative of $\mathbf{u}$
4. *non-homogeneous* ($\mathcal{F} \neq 0$) vs. *homogeneous* ($\mathcal{F} = 0$)

As an example, the time-dependent three-dimensional Navier Stokes equations are a set of four equations. The first three equations, enforcing a momentum (force) balance are homogeneous, non-linear, partial differential equations of second order parametrized by $\mathcal{P} = \begin{pmatrix} Re \end{pmatrix}$. The fourth equation ensures mass conservation and is a homogeneous, linear, partial differential equation of first order.

$$\begin{cases} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \frac{\partial p}{\partial x} - \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) = 0 \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \frac{\partial p}{\partial y} - \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}\right) = 0 \\ \frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \frac{\partial p}{\partial z} - \frac{1}{Re}\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}\right) = 0 \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \end{cases} \tag{2.2}$$

2.1.3 Initial and Boundary Conditions

The solution of (2.2) contains unspecified constants and is therefore called a *general* solution. For example, given that $\mathbf{u}_{gen}(x, y, z, t)$ satisfies equation (2.2), $\tilde{\mathbf{u}}(x, y, z, t) = \mathbf{u}_{gen}(x, y, z, t) + \mathbf{C}$ will satisfy (2.2) as well. In a *particular* solution, all arising constants are specified by numeric values [20].

To obtain numeric values for all entries in C, for each independent variable, as many additional conditions as that variable's highest order in the differential equation need to be supplemented. These conditions are either initial conditions (IC, for time) or boundary conditions (BC, for space). For a general domain of $\begin{pmatrix} x & y & z \end{pmatrix} \in \Omega$ and $t \in [0, T]$ with $0 < T \in \mathbb{R}$, these conditions read as follows:

$$\begin{cases} \mathbf{u}(x, y, z, 0) = \mathbf{u}_0(x, y, z) & \begin{pmatrix} x & y & z \end{pmatrix} \in \Omega & \text{IC} \\ \mathbf{u}(x, y, z, t) = \mathbf{b}(x, y, z, t) & \begin{pmatrix} x & y & z \end{pmatrix} \in \partial\Omega \times t \in [0, T] & \text{BC} \end{cases} \tag{2.3}$$

For the Navier Stokes equations, an initial condition at $t = 0$ and boundary conditions for each u, v, w at all spatial boundaries $\partial\Omega$ need to be added to (2.2) to obtain a particular solution. In contrast, the highest derivative of p in space is of order 1, therefore one boundary condition for p is sufficient to obtain the exact pressure field. These conditions may appear in three different forms:

1. Dirichlet condition: fixed values, e.g. $\mathbf{u}(x, y, z, t) = \mathbf{b}(x, y, z, t)$
2. Neumann condition: fixed derivative, $\frac{\partial \mathbf{u}(x,y,z,t)}{\partial x} = \mathbf{b}(x, y, z, t)$
3. Robin condition: mix of Dirichlet and Neumann condition, e.g. $\mathbf{u}(x, y, z, t) + \frac{\partial \mathbf{u}(x,y,z,t)}{\partial x} = \mathbf{b}(x, y, z, t)$

2.1.4 Boundary-Initial Value Problems

Combining a differential equation in (2.1) with initial and boundary conditions (2.3) yields a boundary-initial value problem (BIVP)

$$\begin{cases} \mathbf{u}_t + \mathcal{N}_{\mathcal{P}}(\mathbf{u}, x, y, z, t) = \mathcal{F}_{\mathcal{P}}(x, y, z, t) & \begin{pmatrix} x & y & z \end{pmatrix} \in \Omega \times t \in [0, T] \\ \mathbf{u}(x, y, z, 0) = \mathbf{u}_0(x, y, z) & \begin{pmatrix} x & y & z \end{pmatrix} \in \Omega \\ \mathbf{u}(x, y, z, t) = \mathbf{b}(x, y, z, t) & \begin{pmatrix} x & y & z \end{pmatrix} \in \partial\Omega \times t \in [0, T] \end{cases} \tag{2.4}$$

whose solution is a particular solution, both satisfying the differential equation in (2.1) and the initial and boundary conditions in (2.3). On the one side, if a solely temporal problem is considered, the corresponding problem lacks a boundary condition and is called an initial value problem (IVP) or Cauchy problem. On the other hand, if a steady-state solution is sought $\left(\frac{\mathbf{u}(x,y,z,t)}{\partial t} = 0\right)$, no initial condition needs to be supplemented and the problem formulation results in a boundary value problem (BVP).

2.1.5 Numerical and Analytic Solutions

Almost all practically relevant differential equations are hard to solve, partially due to strong non-linearities and complex geometrical domains. Numerical approaches that solve differential equations approximately have been researched extensively for decades and with remarkable success. Yet, the computational cost/effort still is a bottleneck in practice, especially for large-scale modeling in space and time, e.g. climate prediction. Therefore, efficient algorithms to solve differential equations both accurately and fast, are of major interest for the scientific computing community. The solution of Navier-Stokes equations, describing fluid flow, have even been posed as one of seven millennium price problems in mathematics.

Analytic solutions only exist for easy and often artificially constructed problems. Despite their practical irrelevance, they can serve as a reference when it comes to evaluating the accuracy of numerical algorithms. This reason also motivates the use of analytic reference solutions in this work.

2.2 Deep Learning and Neural Networks

2.2.1 Introduction

Deep learning is a rapidly emerging discipline that has shown promising results across a diverse range of applications. Even though its beginnings date back to the mid-of 20th century, deep learning has recently enjoyed a resurgence, driven by the availability of "big" data, powerful hardware in the form of graphics processing units (GPUs) as well as more complex (deeper) models [3]. Artificial neural networks (ANNs), which form the basis of deep learning, will now be explained in detail to provide a mathematically grounded framework for this work.

2.2.2 Problem Statement

Suppose that we have given a labeled data set $\mathcal{D} = \{(\mathbf{x}_i, \mathbf{y}_i)\}_{i=1}^N$ of N data points. Then the goal of training an ANN is to determine its parameters $\mathcal{W}$ such that it approximates the mapping inherent to $\mathcal{D}$ as good as possible with respect to some metric $\mathcal{L}$ called loss/objective/energy function

$$\mathcal{W}^* = argmin_{\mathcal{W}} \mathcal{L}(\mathbf{y}, \hat{u}(\mathbf{x})) \tag{2.5}$$

where $\hat{\mathbf{u}}(\mathbf{x})$ is the network's prediction an $\mathbf{y}$ is the label to be approximated. The most common loss function, also employed in this work is the mean squared loss, given by

$$\mathcal{L}(\mathbf{y}, \hat{\mathbf{u}}(\mathbf{x})) = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{y}_i - \hat{\mathbf{u}}(\mathbf{x}_i))^2 \tag{2.6}$$

2.2.3 Universal Approximation Theorem

The reason that (2.5) is a feasible task has been proven by [21]: in theory, a fully-connected ANN can approximate any function to any degree of accuracy. Stated differently, ANNs are nothing but function approximators. In practice, limits may be reached due to insufficient approximation capacity of too small networks, hence too few parameters and a low-dimensional $\mathcal{W}$, and convergence issues in the optimization task of (2.5). The property of ANNs being universal function approximators justifies their use for approximating any quantity of interest such as physically relevant quantities studied later on.

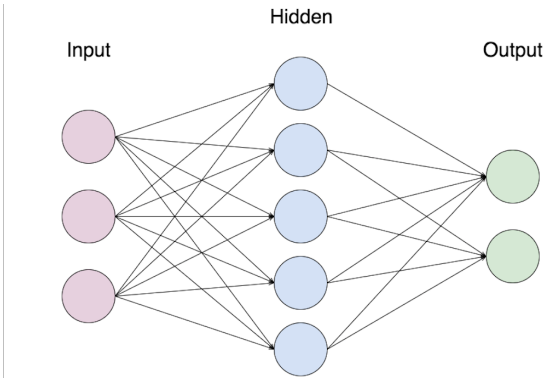


Figure 1: Artificial neural network of depth $D = 2$ with $I = 3$ input units, $D - 1 = 1$ hidden layer with $H = 5$ hidden neurons and $O = 2$ output units.

2.2.4 Architecture of Fully-Connected Networks

There exist several types of network architectures, including fully-connected, convolutional and recurrent neural networks. Even though convolutional and recurrent neural networks have achieved tremendous success in processing images [1] and speech [2], respectively, fully-connected networks will be the only architecture used in this study in line with several other studies in the field of physics-informed machine learning [9, 5, 6, 7, 22, 13, 23, 14, 15, 16, 17, 18, 24, 11, 12, 10, 25].

A fully-connected ANN of depth $D \in \mathbb{R}$ is a layered structure of neurons that each contain a numeric value and process information from layer to layer (figure 1). Raw data $\mathbf{x}^0 \in \mathbb{R}^I$ is fed into an input layer with $I \in \mathbb{R}$ neurons, which pass the information recursively to $D - 1$ hidden layers with each $H \in \mathbb{R}$ neurons. The network's output with $O \in \mathbb{R}$ units contains the predicted quantity of interest, $\hat{\mathbf{u}}(\mathbf{x})$. By convention, any network with at least two layers $D \geq 2$ (and therefore at least one hidden layer $D-1 \geq 1$) is called deep.

2.2.5 Forward Pass

The process of inferring the quantities of interest in the output neurons from raw data in the input units is called forward pass. It is the evaluation of an analytic closed-form function that is a concatenation of linear and non-linear operations. Within each layer, the d^j-dimensional input from the preceding layer $\mathbf{x}^j$ is subjected to an affine transformation

$$A^{j+1}(\mathbf{x}^j) = \mathbf{W}^{j,j+1}\mathbf{x}^j + \mathbf{b}^{j+1} \tag{2.7}$$

where $\mathbf{W}^{j,j+1} \in \mathbb{R}^{d^j \times d^{j+1}}$ is the weight matrix connecting layers j and $j+1$ and $\mathbf{b}^{j+1} \in \mathbb{R}^{d^{j+1}}$ is the bias vector. Subsequently, any non-linear activation function $\sigma()$ is

applied neuron-wise and $\sigma(A^{j+1}(\mathbf{x}^j)) \in \mathbb{R}^{d^{j+1}}$ is fed into the next layer. By recursively performing this transformation in a D-layer neural network, the mapping between input $\mathbf{x}^0 \in \mathbb{R}^I$ and output $\hat{\mathbf{u}}(\mathbf{x}^0) \in \mathbb{R}^O$ becomes

$$\hat{\mathbf{u}}(\mathbf{x}^0) = (A_D \circ \sigma \circ A_{D-1} \circ \ldots \circ \sigma \circ A_1)(\mathbf{x}^0) \tag{2.8}$$

Note that $d^0 = I$, $d^D = O$ and $d^k = H$ for $k = 1, ..., D-1$. Common non-linear activation functions include the hyperbolic tangent function,

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \in [-1, 1] \tag{2.9}$$

which is applied neuron-wise and used throughout this work. The process of computing the weights and biases $\mathcal{W}^* = \{\mathbf{W}^{j,j+1}, \mathbf{b}^{j+1}\}_{j=0}^{D}$ of an ANN is called training which is explained next.

2.2.6 Backward Pass

The optimization task presented in (2.5) is, in almost all applications, high-dimensional with a number of parameters in the range of hundreds, thousands or possibly millions. Therefore, training an ANN is a critical and computationally expensive task. Many optimizers are gradient-based, i.e. they require the gradients of the loss functions with respect to all parameters $\mathcal{W}$. We have seen in (2.8) that the network output is a concatenation of many functions containing the input and all network parameters. The gradients are hence computed by applying the chain rule from the output, layer by layer, to the input. This flow of passing gradients is backward, giving the name backward pass or backpropagation. This is in strong contrast to a network's forward pass for predictions, where the flow of information is from input to output layer. Two common optimizers are explained in detail below.

Adam Optimizer The idea behind adaptive momentum estimation (Adam) is to combine two concepts [26]:

1. momentum: accumulate gradients over time to incorporate an exponentially-weighted average of the gradient (first moment)

2. damp oscillations for high-variance directions (second moment)

For the set of hyperparameters $(\beta_1, \beta_2, \epsilon)$ with standard values $(0.9, 0.999, 10^{-8})$, the algorithm looks as follows:

$$
\begin{aligned}
\mathbf{m}^{k+1} &= \beta_1 \mathbf{m}^k + (1-\beta_1)\nabla_{\mathcal{W}}\mathcal{L}(\mathcal{W}^k) \\
\mathbf{v}^{k+1} &= \beta_2 \mathbf{v}^k + (1-\beta_2)(\nabla_{\mathcal{W}}\mathcal{L}(\mathcal{W}^k) \circ \nabla_{\mathcal{W}}\mathcal{L}(\mathcal{W}^k)) \\
\hat{\mathbf{m}}^{k+1} &= \frac{\mathbf{m}^{k+1}}{1-\beta_1^{k+1}} \\
\hat{\mathbf{v}}^{k+1} &= \frac{\mathbf{v}^{k+1}}{1-\beta_2^{k+1}} \\
\mathcal{W}^{k+1} &= \mathcal{W}^k - \alpha \frac{\hat{\mathbf{m}}^{k+1}}{\sqrt{\hat{\mathbf{v}}^{k+1}} + \epsilon}
\end{aligned}
\tag{2.10}
$$

where $\mathbf{m}^0 = \mathbf{v}^0 = \mathbf{0}$, α is called step size (learning rate), all operations are performed element-wise and the superscript denotes the $k-$th iteration during optimization.

L-BFGS Optimizer The limited memory Broyden-Fletcher-Goldfarb-Shanno algorithm [27] belongs to the family of quasi-Newton methods and uses an estimate of the inverse Hessian matrix $\mathbf{H}^{-1}$ to steer its search through parameter space:

$$
\mathcal{W}^{k+1} = \mathcal{W}^k - \mathbf{H}^{-1}\nabla_{\mathcal{W}}\mathcal{L}(\mathcal{W}^k) \tag{2.11}
$$

Remark $\nabla_{\mathcal{W}}\mathcal{L}(\mathcal{W}^k)$ is usually computed based on $\mathcal{D}_{mini-batch} \subset \mathcal{D}$, a subset of the provided database, called mini-batch. The resulting gradient is therefore an estimate of the actual gradient with respect to the entire dataset $\mathcal{D}$, resulting in a stochastic optimization process. As the computation of $\nabla_{\mathcal{W}}\mathcal{L}(\mathcal{W}^k)$ is linear in the size of the mini-batch, smaller mini-batches yield faster gradient updates. As long as $\mathcal{D}_{mini-batch}$ is representative of $\mathcal{D}$, the gradient updates will be close to full-batch updates. While the Adam optimizer works on user-specified mini-batches, L-BFGS is a full-batch optimizer. The number of iterations for the Adam optimizer is usually given by epochs, with one epoch corresponding to one run through the entire data set $\mathcal{D}$. Therefore, one epoch consists of $\approx \frac{|\mathcal{D}|}{|\mathcal{D}_{mini-batch}|}$ mini-batch updates with $|\mathcal{D}|$ denoting the number of data points in $\mathcal{D}$.

2.2.7 Xavier-/Glorot Initialization

As the optimization steps provided by Adam (2.10) and L-BFGS (2.11) have only defined an iterative process, some initial values for $\mathcal{W}$ need to be supplemented. Initialization is of primary importance in this work as transfer learning is simply a particular form of initialization. The initialization scheme named after Xavier Glorot ensures that the variance of input and output are in a similar range, leading to smooth gradient updates

[19]. To do so, all bias vectors $\mathbf{b}^j$ are initialized as zero vectors and the weight matrices according to

$$(\mathbf{W}^{j,j+1})_{mn} \sim \mathcal{U}\Big[-\frac{1}{\sqrt{d^j}}, \frac{1}{\sqrt{d^j}}\Big] \tag{2.12}$$

where the subscript mn denotes the matrix index in row m and column n and $\mathcal{U}[-a, a]$ is the uniform distribution on the interval $(-a, a)$.

2.3 Physics-Informed Neural Networks (PINNs)

2.3.1 Introduction

PINNs are a deep learning framework for solving differential equations numerically. The pioneering idea of solving BIVPs with ANNs dates back to 1997 [9]. There, Lagaris et al. provide a mathematical framework in which the solution of a differential equation is approximated by a single hidden layer ANN. All gradient computations and optimization steps had to be calculated analytically as no deep learning libraries had been developed by that time. As part of the recent rise of deep learning, several libraries including TensorFlow [28] and PyTorch [29] have been developed and are widely used in the machine learning community. This development has motivated the idea of using deeper networks for solving differential equations [5, 6, 7] and general deep learning frameworks, called *physics-informed neural networks*, have been created. Automatic differentiation, a generalized method of backpropagation for efficient and accurate computation of derivatives [30], can be readily used in multi-layer networks without the need to compute derivatives by hand. The ease of computation and implementation has led to a large number of publications on PINNs in the past years. In the following, the two preceding chapters 2.1 and 2.2 will be combined and synthesized to introduce the general framework of PINNs and summarize recent advances in the field.

2.3.2 General Framework

In the seminal work of Raissi et al. [5], the numerical solution of BIVPs has been formulated as an optimization problem. The idea is to incorporate the governing equations of (2.4) into the optimization problem (2.5) with loss function (2.6):

$$\mathcal{L} = \underbrace{\mathcal{L}_{de}}_{\text{differential equation}} + \underbrace{w_i\mathcal{L}_i}_{\text{initial condition}} + \underbrace{w_b\mathcal{L}_b}_{\text{boundary condition}} \tag{2.13}$$

where each term accounts for one equation of the BIVP formulated in (2.4). The scalars (w_i, w_b) introduce a weighting to account for the relative importance of each loss term. All spatio-temporal variables $\mathbf{x}0 = \begin{pmatrix} x & y & z & t \end{pmatrix}$ serve as input parameters and the physical quantity of interest is the network output, $\hat{\mathbf{u}}(\mathbf{x}^0) = \begin{pmatrix} \hat{u} & \hat{v} & \hat{w} & \hat{p} \end{pmatrix}$ with velocities u, v, w and pressure p. Depending on the problem, the quantities of interest may change, but the mentioned ones are required for this work. To incorporate the differential equation, automatic differentiation is employed to compute the necessary derivatives of the output variables with respect to the input variables. Then, the loss function in (2.13) takes the form

$$\mathcal{L} = \frac{1}{N_{de}} \sum_{j=1}^{N_{de}} |\mathbf{f}(\mathbf{x}_{de}^j)|^2 + \frac{w_i}{N_i} \sum_{j=1}^{N_i} |\hat{\mathbf{u}}(\mathbf{x}_i^j) - \mathbf{y}_i^j|^2 + \frac{w_b}{N_b} \sum_{i=1}^{N_b} |\hat{\mathbf{u}}(\mathbf{x}_b^j) - \mathbf{y}_b^j|^2 \tag{2.14}$$

where $\mathbf{f}(\mathbf{x}) := \mathbf{u}_t + \mathcal{N}_{\mathcal{P}}(\mathbf{u}, x, y, z, t) - \mathcal{F}_{\mathcal{P}}(x, y, z, t)$ denotes the differential equation residual and $\mathcal{D}_{de} = \{(\mathbf{x}_{de}^j)\}_{j=1}^{N_{de}}$, $\mathcal{D}_i = \{(\mathbf{x}_i^j, \mathbf{u}_i^j)\}_{j=1}^{N_i}$ and $\mathcal{D}_b = \{(\mathbf{x}_b^j, \mathbf{u}_b^j)\}_{j=1}^{N_b}$ contain residual, initial and boundary data, respectively.

2.3.3 Theoretical Guarantees

Most of the research on PINNs has been based on empirical findings without mathematically grounded theorems. For example, getting stuck in local minima instead of global minima during optimization is a common problem in deep learning [19]. There is no theoretical guarantee, regardless of the network architecture or BIVP, which degree of accuracy the converged solution satisfies, possibly due to local minima [5]. It is particularly important to point out that the error in the loss function $\mathcal{L}$ measures the degree to which the differential equation is satisfied, not the error of the solution $\mathbf{u}$ itself with respect to an analytical or reference solution. Therefore, a small loss value for $\mathcal{L}$ does not necessarily imply an accurate solution [18]. The required loss value $\mathcal{L}$ is, for now, impossible to determine starting from a specific solution accuracy in terms of $\mathbf{u}$ and makes the effect of optimization less predictable. In line with several other works on PINNs, the present study will be based on empirical findings but acknowledges the need for theoretical results.

2.3.4 Enforcement of Initial and Boundary Conditions

There are two distinct possibilities to encode initial and boundary conditions:

On the one side, to account for each as a separate term in the loss function as in (2.14). The goal of this enforcement as *soft constraints* is to optimize multiple objectives at once, i.e. to minimize each loss term simultaneously.

On the other side, it is possible to transform the network output to some function of the form

$$\Psi(\mathbf{x}^0) = \mathbf{h}(\mathbf{x}) + d(\mathbf{x}) \cdot \hat{\mathbf{u}}(\mathbf{x}^0) \tag{2.15}$$

where $\mathbf{h}(\mathbf{x})$ accounts for the initial and boundary conditions and $d(\mathbf{x})$ is some function whose value correlates with the distance to the spatio-temporal domain's boundary. This encoding of initial and boundary conditions is referred to as *hard constraints*. Depending on the complexity of the geometry, there exist two approaches to enforce initial and boundary conditions as hard constraints:

Simple Geometries $\mathbf{h}(\mathbf{x})$ and $d(\mathbf{x})$ can be chosen such that (2.15) satisfies the initial and boundary data by construction. For example, on an exemplary domain $x \in [-1, 1]$ with boundary conditions $u(-1) = u(1) = 2$, $\Psi(\mathbf{x}^0) = 2 + (1 - x^2) \cdot \hat{\mathbf{u}}(\mathbf{x}^0)$ leads to perfect accuracy at the boundary [9].

Complex Geometries The analytic expressions of $\mathbf{h}(\mathbf{x})$ and $d(\mathbf{x})$ cannot be manufactured by hand but need to be approximated by neural networks. The computational effort compared to the PINN itself is low, as a small network architecture with few parameters is sufficient for the two required networks $\mathbf{h}(\mathbf{x})$ and $d(\mathbf{x})$ [25].

2.3.5 Unbalanced Gradients and Adaptive Weights

If initial and boundary conditions are enforced as soft constraints, the problem in (2.5) becomes a multi-objective optimization task with several terms in (2.13). It has been observed across various BIVPs that the accuracy of some terms might be sacrificed for the sake of others [17, 18]. More often than not, boundary and initial conditions had not been satisfied accurately. Instead, optimization was primarily focused on the differential equation residual. Wang et al. [17] have identified an underlying fundamental mode of failure: the gradients of each loss term in (2.13) are imbalanced, i.e. the gradients of some loss term (often initial/boundary term) are sharply centered around zero and have significantly smaller values than others (often differential equation term). This leads to tiny updates with respect to the gradients of terms with small gradients in (2.10), (2.11). The resulting accuracy of such terms is often inferior at the end of optimization.

One might naturally come up with the idea of adjusting the weighting scalars (w_i, w_b) in (2.13). The naive way would be to try out several configurations by hand, observe the accuracy of each loss terms and adjust the weighting accordingly. In the end, one could choose the weighting which yielded best overall accuracy of the predicted solution. Unfortunately, this process is laborious, time-consuming and not generalizable to other problem configurations.

Instead, Wang et al. have proposed an adaptive weighting strategy, called *learning rate annealing*, that adjusts the scalars in (w_i, w_b) dynamically during model training. Their algorithm is inspired by updates of the learning rate of the Adam optimizer and uses gradient statistics to balance the interplay between different loss terms. Across several benchmark problems, including the Helmholtz, Poisson and Klein-Gordon equations, their adaptive weighting strategy leads to noticeable improvements [17].

2.3.6 Employed Optimization Strategy

In this bachelor's thesis, initial and boundary conditions will be enforced as soft constraints to circumvent the construction of possibly laborious analytic transformations (simple geometries) or additional networks for spatio-temporal boundaries and distance functions (complex geometries). Regardless of the geometry complexity, a multi-objective loss function in (2.13) is optimized, therefore presenting a general framework. The issue of gradient pathologies [17] is recognized, but all examples studied in this bachelor's thesis are taken from previous publications that investigated their solution with PINNs. Their optimization approach as well as the employed fixed or dynamic weighting strategy for (w_i, w_b) will be adopted.

3 Transfer Learning with Physics-Informed Neural Networks

3.1 Introduction to Transfer Learning

In transfer learning, knowledge acquired from some task is exploited to improve the learning of a related, but different task [31]. Transfer learning in a computational setting is motivated by human learners, who automatically and unconsciously recognize similar tasks and apply experience from previous problems to new ones. While the concept of transfer learning in a computational setting is motivated and inspired by its human counterpart, there is no ambition to recreate it due to our limited understanding of human transfer learning.

3.2 Terminology

Let us formalize the concept of transfer learning by introducing a *source task*. The knowledge acquired by some model that is trained on it will be transferred to a related, but not identical *target task*. In the context of this work, the model is a physics-informed neural network and the knowledge is accumulated in the set of parameters and weights $\mathcal{W}$ that are learned during model training (2.10), (2.11). Two tasks are related, if the problem setup, described by the physical parameter vector $\mathcal{P}$, is similar. $\mathcal{P}$ contains a characteristic quantity of the problem, e.g. Reynolds number that is characteristic for flow problems [32]. Transfer learning in the context of physics-informed neural networks may also be interpreted as a multi-stage training process (figure 2).

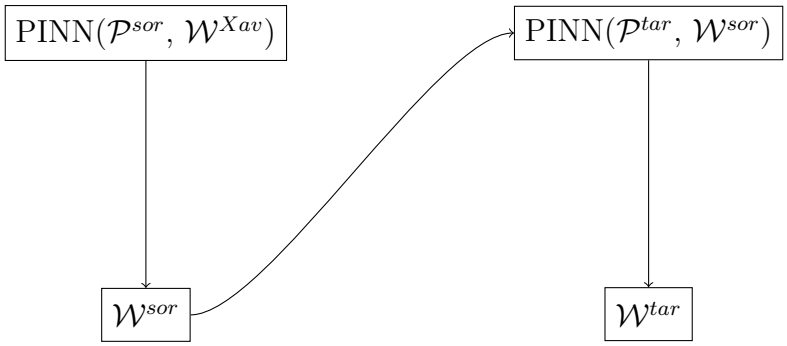


Figure 2: A source PINN (top left), tasked to solve a BIVP with physical configuration $\mathcal{P}^{sor}$ and Xavier initialization $\mathcal{W}^{Xav}$, is trained and the weight vector $\mathcal{W}^{sor}$ (bottom left) is stored. A new PINN (top right), tasked to solve a BIVP with physical configuration $\mathcal{P}^{tar}$ and reference initialization $\mathcal{W}^{sor}$ is trained and the weights become $\mathcal{W}^{tar}$.

3.3 Metrics for Evaluation

There are three common ways in which transfer learning might improve training [31]:

First, the *initial performance* of a trained source model on a target task. That is, how do the network parameters $\mathcal{W}^{sor}$ that succeed on some source task $\mathcal{P}^{sor}$ perform on the new target task $\mathcal{P}^{tar}$ before any training is done?

Second, the *speed of convergence*. This metric will be evaluated based on the amount of time it takes to reach a certain level of accuracy.

And third, the *final performance* of the trained model on the target task. That is, how do the network parameters $\mathcal{W}^{tar}$ after training perform on the new target task $\mathcal{P}^{tar}$?

For the initial and final performance measure, two metrics are possible: the value of the loss in (2.14) and the relative $\mathcal{L}_2$ error with respect to a reference solution. This reference $\mathbf{u}(\mathbf{x})$ may be in the form of either a closed-form analytic solution or a high-fidelity numerical simulation of another algorithm. Then, the relative $\mathcal{L}_2$ error is defined as

$$\mathcal{L}_2(\mathbf{u}(\mathbf{x},t),\hat{\mathbf{u}}(\mathbf{x},t)) = \frac{||\mathbf{u}(\mathbf{x},t)-\hat{\mathbf{u}}(\mathbf{x},t)||_2}{||\mathbf{u}(\mathbf{x},t)||_2} \tag{3.1}$$

where $||\cdot||_2$ denotes the standard L_2 norm.

3.3.1 Expectations

It is expected that both initial and final performance of source model initialization should outperform its random counterpart for similar problems.

Moreover, PINNs should converge faster to an accurate solution for a source model initialization than random initialization. This hypothesis is based on the assumption that the initial performance with source model initialization is better and hence the required improvement to reach a certain level of accuracy is smaller than for random initialization.

In an ideal scenario, these three metrics could behave as shown in figure 3 and would improve the training process significantly.

3.4 Literature Review

Several works have applied transfer learning to physics-informed neural networks and ultimately sparked the idea behind this thesis. The aim of this chapter is to summarize their transfer learning approaches, numerical results and interpretations.

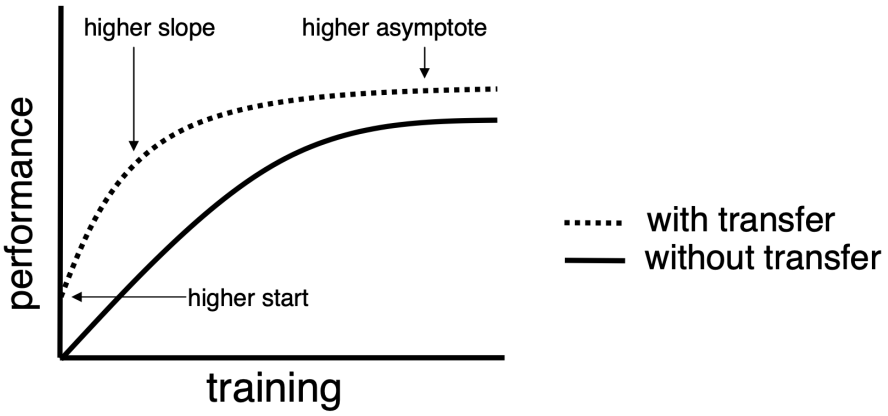


Figure 3: Exemplary performance curves for source model (dashed line) and random (solid line) initialization. In a perfect case, transfer learning would improve the initial performance (higher start), speed of convergence (higher slope) and final performance (higher asymptote). Picture taken from [31].

3.4.1 Transfer Learning in Solid Mechanics

In [13], Haghighat et al. have studied physics-informed neural networks on linear and non-linear elasticity problems from continuum mechanics. For a two-dimensional example of the linear elasticity of a plate, they have measurement data available for several parameter configurations of $\mathcal{P} = \Big(\lambda\Big)$, where the Lamé parameter λ is characteristic for the problem from a physical point of view. They pre-train a source model on the material parameter $\mathcal{P}^{tar} = \Big(0.5\Big)$ and store the weights and biases $\mathcal{W}^{sor}$ of the trained model. Then, they train four target models with $\mathcal{P}_1^{tar} = \Big(0.1\Big)$, $\mathcal{P}_2^{tar} = \Big(1.0\Big)$, $\mathcal{P}_3^{tar} = \Big(1.5\Big)$, $\mathcal{P}_4^{tar} = \Big(2.0\Big)$ and use the trained source model as initialization. They have reported vastly accelerated convergence and source model initialization outperforms its random counterpart in all three metrics, namely initial performance, speed of convergence and final performance. Looking at the performance/loss curves in [13], one can observe that the source model initialization is particularly effective for $\lambda \in \{0.1, 1.0, 1.5\}$ and less effective for $\lambda = 2.0$. This might be due to the fact that the task presented by $\lambda = 2.0$ is too different from $\lambda = 0.5$ and the tasks corresponding to $\lambda \in \{0.1, 1.0, 1.5\}$ are more related to $\lambda = 0.5$, thus yielding better improvements from source model initialization. The validity of this last hypothesis shouldn't be overemphasized as it is only based on one example, but it will be kept in mind for the following chapters.

3.4.2 Transfer Learning for Multiscale Flows

In [23], Lou et al. have applied physics-informed neural networks to multiscale flows by employing the Boltzmann-Bhatnagar-Gross-Krook model [33]. They have studied a forward setting of the Kovasznay flow [34] and trained a source model on a Reynolds number $\mathcal{P} = \Big(Re\Big)$ with $\mathcal{P}^{sor} = \Big(10\Big)$. Then, they defined three new physical target settings with $\mathcal{P}_1^{tar} = \Big(20\Big)$, $\mathcal{P}_2^{tar} = \Big(40\Big)$, $\mathcal{P}_3^{tar} = \Big(60\Big)$ and compared the effectiveness of transfer learning from two perspectives:

First, they trained the target model with source and random initialization by employing the L-BFGS-B optimizer only. In this case, while the computational effort was similar, the physics-informed neural network with source initialization outperformed its random counterpart by up to two orders of magnitude in the relative $\mathcal{L}_2$ norm. They claim that an educated instead of a random guess is crucial for the L-BFGS-B optimizer in order to overcome bad local minima.

Second, they compared computational effort for two similarly accurate solutions. To do so, Lou et al. trained the model with random initialization on the Adam optimizer and L-BFGS-B subsequently, while the network with source initialization was trained with the L-BFGS-B optimizer only. The longer optimization process in the former case could compensate its worse initialization and achieve comparable accuracy. Yet, the computational effort measured in terms of time was about three times as high.

In summary, transfer learning led either to a three-fold speed-up with comparable accuracy or to a two orders of magnitude better accuracy with comparable computational effort, underlining the effectiveness of transfer learning for physics-informed neural networks.

3.4.3 Transfer Learning for Navier Stokes Equations

In the publication on NSF nets [12], the authors have studied a transfer learning setting for the Kovasznay flow, similar to the approach outlined in section 3.4.2. They have pre-trained a model on $\mathcal{P} = \Big(Re\Big)$ with $\mathcal{P}^{sor} = \Big(40\Big)$ and transferred the trained parameters $\mathcal{W}^{sor}$ to the problem $\mathcal{P}^{tar} = \Big(60\Big)$. The target model has then been trained on L-BFGS optimizer with random and source initialization, while the initialization from a pre-trained source model outperformed its random counterpart in both accuracy and computational efficiency (speed of convergence): while the computational efficiency is reduced by a factor of ≈ 5, accuracy is improved by more than one order of magnitude for each velocity component and the pressure in the relative $\mathcal{L}_2$ norm.

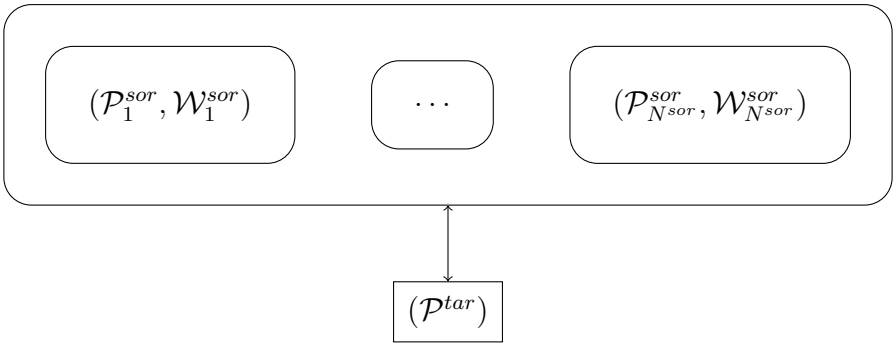


Figure 4: Database of trained physics-informed neural networks $\mathcal{D}$. Each entry in the dictionary-like database is referred to by a key $\mathcal{P}_i^{sor}$ describing the physical problem setup and contains the trained network's parameters $\mathcal{W}_i^{sor}$.

3.5 Databases of Physics-Informed Neural Networks

3.5.1 Idea Sketch

Inspired by the previous three transfer learning strategies (section 3.4), this work proposes a new paradigm in training physics-informed neural networks. The idea is to build databases of trained physics-informed neural networks and exploit this database for smart initialization.

More specifically, the database can be interpreted as a dictionary $(\mathcal{P}^{sor}, \mathcal{W}^{sor})$. There, the key $\mathcal{P}^{sor}$ is the physical parameter vector characterizing the problem. The corresponding value $\mathcal{W}^{sor}$ contains a network's parameters, i.e. weights and biases. This database containing N^{sor} entries will be denoted by $\mathcal{D} = \{(\mathcal{P}_i^{sor}, \mathcal{W}_i^{sor})\}_{i=1}^{N^{sor}}$ with $|\mathcal{D}| = N^{sor}$ (figure 4).

3.5.2 Similarity Measurement

The fundamental question becomes how to exploit such a database with the goal of accelerating training of physics-informed neural networks. As observed in [13] and explained in section 3.4.1, the effectiveness of transfer learning increased with problem similarity. This motivates the approach to initialize the target model with a trained physics-informed neural network of which the problem statement is most similar. With a similarity measurement $\mathcal{S}(\mathcal{P}_1, \mathcal{P}_2)$, attaining high values for similar problem setups, an algorithmic

approach is defined below in algorithm 1.

Algorithm 1:

Obtain access to/construct a database of trained physics-informed neural networks $\mathcal{D} = \{(\mathcal{P}_i^{sor}, \mathcal{W}_i^{sor})\}_{i=1}^{N^{sor}}$

Define a new target problem so solve, characterized by $\mathcal{P}^{tar}$

Initialize a vector of similarities $\mathbf{s}$ of size N^{sor} **for** $i \leftarrow 1$ **to** N^{sor} **do**

    Compute the similarity measurement between target task and the i-th database entry $\mathcal{S}(\mathcal{P}_i^{sor}, \mathcal{P}^{tar})$

    Store the similarity $\mathbf{s}(i) \leftarrow \mathcal{S}(\mathcal{P}_i^{sor}, \mathcal{P}^{tar})$

end

Determine the database entry with highest similarity: $s^* = argmax_{i=1,\dots,N^{sor}} \mathbf{s}(i)$

Train the target task with source initialization of the physics-informed neural network s^*: $(\mathcal{P}^{tar}, \mathcal{W}_{s^*}^{tar})$

Finally, an analytic computation of the similarity measurement is required:

$$\mathcal{S}(\mathcal{P}_i^{sor}, \mathcal{P}^{tar}) = 1 - \frac{||\mathcal{P}_i^{sor} - \mathcal{P}^{tar}||}{||\mathcal{P}^{tar}||} \qquad \in (-\infty, +1] \tag{3.2}$$

with

$$\mathcal{S}(\mathcal{P}_i^{sor}, \mathcal{P}^{tar}) \begin{cases} \to 1 & \text{identical physical problems} \\ \to -\infty & \text{unrelated physical problems} \end{cases} \tag{3.3}$$

3.6 Implementation Details: DeepXDE

DeepXDE [8] is a deep learning library built on top of TensorFlow [28]. The open-source software package is a flexible tool to implement PINNs and provides all necessary functionality for transfer learning, therefore used in this bachelor's thesis. All commands closely resemble the mathematical formulation of a BIVP, making it an easy-to-use tool.

All code files in Python accompanying this manuscript are publicly available on GitHub (`https://github.com/mariusmerkle/TL-PINNs`). The reader is encouraged to experiment with the code and suggest further improvements or report bugs to the author (marius.merkle@tum.de).

4 Numerical Results

In this section, the effectiveness of the algorithm 1 will be examined for four distinct forward problems, namely the Poisson equation, Burger's equation, Kovasznay and Beltrami flow. The problem setup can be compactly described with the spatio-temporal domain, initial and boundary conditions as well as material parameters.

4.1 Poisson Equation

4.1.1 Boundary-Value Problem

This first part aims to give an intuitive, visual and qualitative introduction to transfer learning. The problem setup is intentionally chosen to be simple such that the reader can focus on understanding the operating principle of transfer learning. Poisson's equation is a second-order differential equation that describes various physical boundary value problems including applications in heat transfer:

$$\begin{cases} \frac{\partial^2 u}{\partial x^2} + \nu\pi^2 sin(\pi x) = 0 \quad x \in [-1, 1] \\ u(x = -1) = 0 \\ u(x = 1) = 0 \end{cases} \tag{4.1}$$

The analytic solution is given by $u(x) = \nu sin(\pi x)$ and therefore describes a simple oscillatory behavior with Dirichlet boundary conditions at both ends. In this scenario, ν is chosen to be the only physical parameter, hence a problem is fully described by $\mathcal{P} = \begin{pmatrix} \nu \end{pmatrix}$ while all other constraints such as the geometry, differential equation and boundary conditions are kept constant.

4.1.2 Database Generation

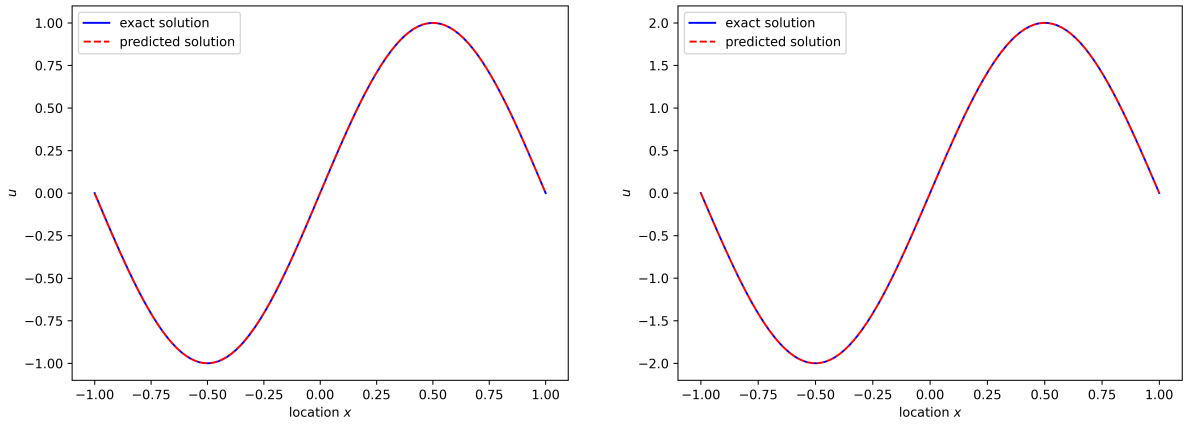
The first step is to generate a database $\mathcal{D}$ in which the neural network weights with corresponding physical parameters for the problem setup are stored. In this case, $N^{sor} = 3$ PINNs corresponding to $\mathcal{P}_1^{sor} = \begin{pmatrix} 1 \end{pmatrix}$, $\mathcal{P}_2^{sor} = \begin{pmatrix} 2 \end{pmatrix}$ and $\mathcal{P}_3^{sor} = \begin{pmatrix} 3 \end{pmatrix}$ are trained and its parameters are stored. Each PINN infers the scalar quantity $\hat{\mathbf{u}} = \begin{pmatrix} \hat{u} \end{pmatrix}$ from a single input neuron $\mathbf{x}^0 = \begin{pmatrix} x \end{pmatrix}$. An exhaustive overview of all hyperparameters and the optimization procedure is given in table 1.

Figure 5 clearly shows that the predicted and exact solution do very well agree. In fact, the network's predictions overshadow the analytical solution in all three cases, i.e. a perfect prediction from a qualitative perspective.

The database $\mathcal{D} = \{(\mathcal{P}_i^{sor}, \mathcal{W}_i^{sor})\}_{i=1}^3$ will serve as a source of initialization in the next section.

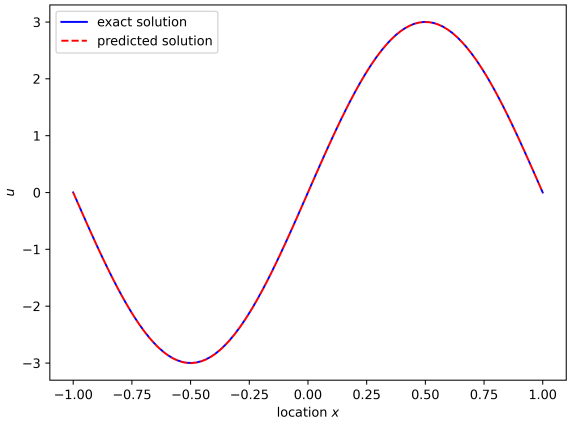
Hidden layers	3
Hidden units	20
Activation function	tanh
Domain points	50
Spatial boundary points	10
Optimizer	Adam, 10.000 epochs

Table 1: Hyperparameter configuration for the steady one-dimensional Poisson equation.



(a) $u(x)$ for $\nu = 1$.

(b) $u(x)$ for $\nu = 2$.



(c) $u(x)$ for $\nu = 3$.

Figure 5: Neural network predictions for the steady one-dimensional Poisson equation. For three different values of ν, $\nu = 1$ (top left), $\nu = 2$ (top right) and $\nu = 3$ (bottom), the predicted solution (dashed red) overshadows the exact solution (solid blue).

4.1.3 Database Initialization

Next, $N^{tar} = 2$ random physical parameter vectors are generated: $\mathcal{P}_1^{tar} = \begin{pmatrix} 1.4 \end{pmatrix}$ and $\mathcal{P}_2^{tar} = \begin{pmatrix} 2.9 \end{pmatrix}$.

$\mathcal{P}_1^{tar} = \begin{pmatrix} 1.4 \end{pmatrix}$ In figure 6, the evolution of the predicted solution for $\mathcal{P}_1^{tar}$ at different stages of optimization is plotted. The left column represents Xavier (random) initialization, while the right column follows algorithm 1.

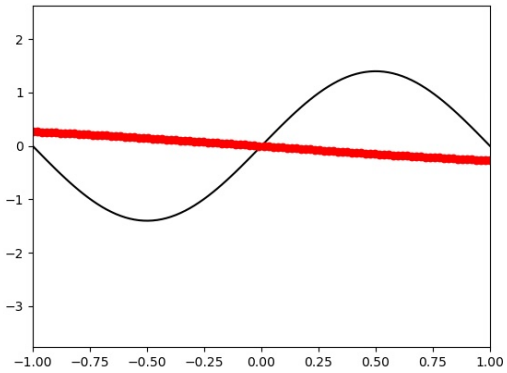
First and foremost, the top row shows the different initializations: while the random strategy results in a straight line (figure 6a), its non-random counterpart picked the trained network solution based on $\mathcal{P}_1^{sor} = \begin{pmatrix} 1 \end{pmatrix}$, as the amplitude is ≈ 1.0 (figure 6b). Therefore, a source initialization from the database here results in an oscillatory behavior that resembles the actual behavior much better than random initialization.

Second, one can observe that the PINN with random initialization slowly approaches the exact solution, but cannot approximate it accurately after 100 epochs (figures 6c, 6e). Meanwhile, the model with database initialization continuously approaches the exact solution better (figure 6d) and almost matches the exact solution after 100 epochs (figure 6f). Note that the underlying similarity was $\mathcal{S}(\nu) = 0.4$ and hence small. Stated differently, there has not been a close-by model (in terms of physical parameters) saved in the database. Still, transfer learning resulted in a remarkable boost of the training process.

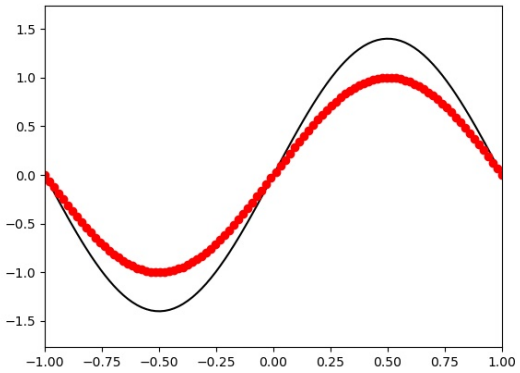
$\mathcal{P}_2^{tar} = \begin{pmatrix} 2.9 \end{pmatrix}$ Again, both training procedures based on random initialization and algorithm 1 are compared. This case is an interesting contrast to the previous as a PINN with close-by physical parameter $\mathcal{P}_3^{tar} = \begin{pmatrix} 3 \end{pmatrix}$ is stored in the database. Figure 7 shows a similar pattern as figure 6: database initialization results in a better initial approximation, higher speed of convergence and more accurate final predictions. In contrast to the previous case, the similarity between $\nu = 2.9$ and the corresponding difference in the database is $\mathcal{S}(\nu) \approx 0.97$ and therefore close to 1 (identifcal physical problem). At the same time, the improvements of transfer learning in figure 7 are even stronger than in figure 6. This observation leads to the hypothesis that a *boost of transfer learning correlates with underlying similarity.*

4.1.4 Summary

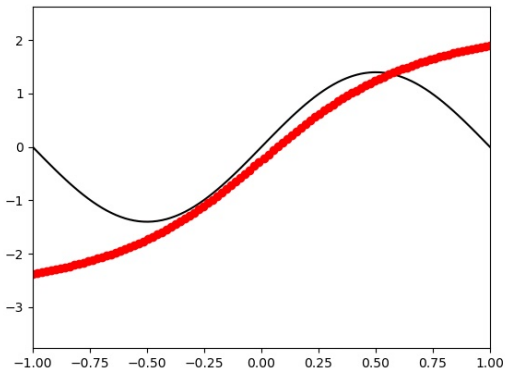
In this section, the proposed approach has been intuitively introduced with the Poisson equation. For three source models in the database, we have seen on two target models that



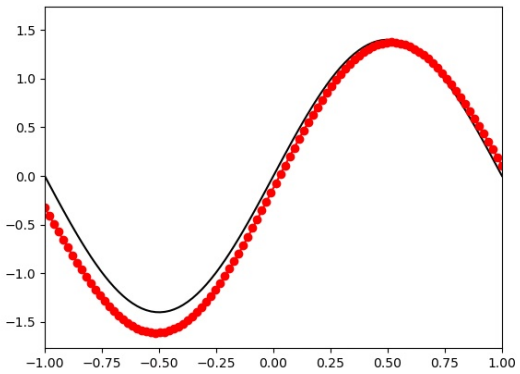
(a) $u(x)$ at initialization.



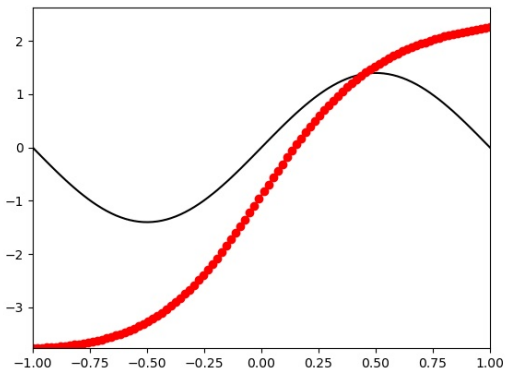
(b) $u(x)$ at initialization.



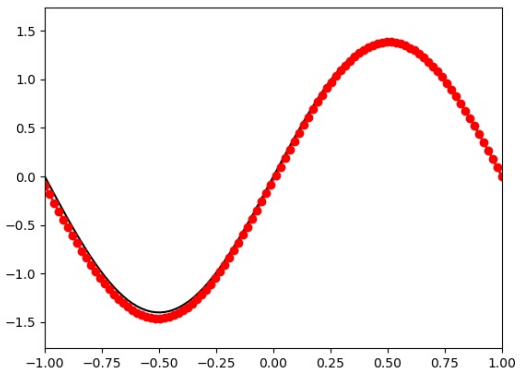
(c) $u(x)$ after 50 epochs.



(d) $u(x)$ after 50 epochs.

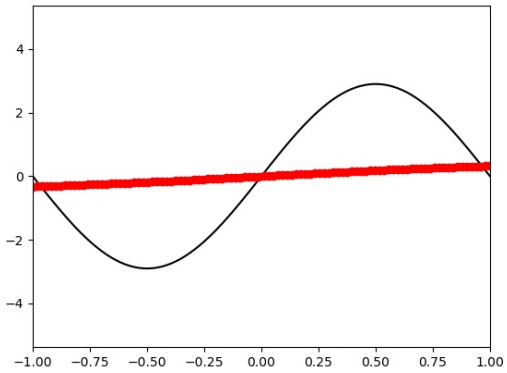


(e) $u(x)$ after 100 epochs.

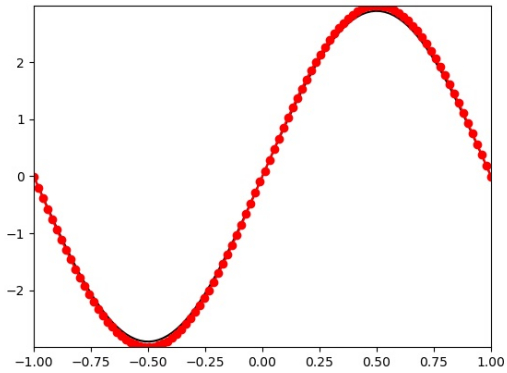


(f) $u(x)$ after 100 epochs.

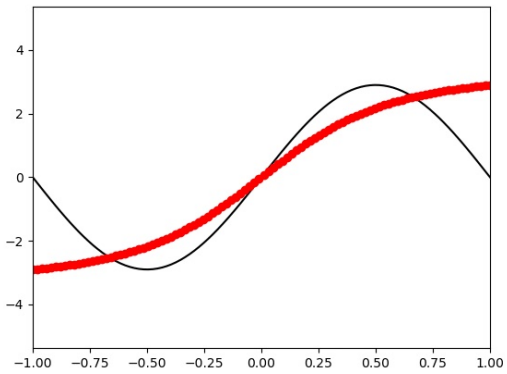
Figure 6: Neural network predictions (red) and the corresponding analytical solution (black) for random (left) and database (right) initialization after 0 (top), 50 (middle) and 100 (bottom) epochs. $\nu = 1.4$.



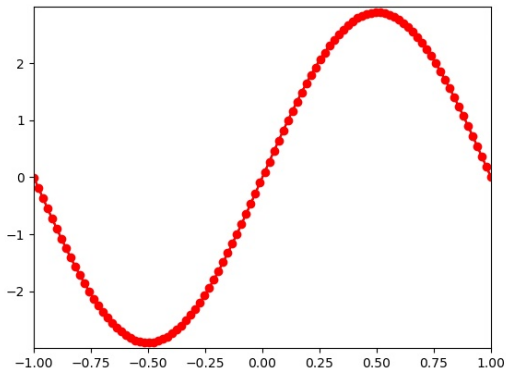
(a) $u(x)$ at initialization.



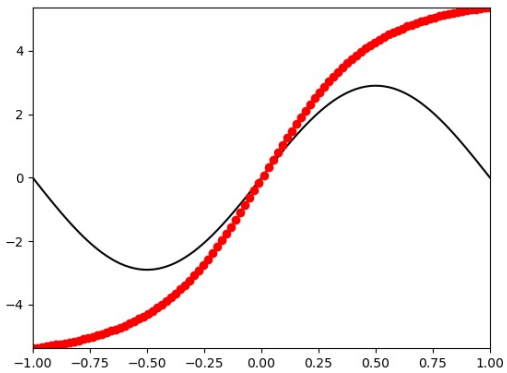
(b) $u(x)$ at initialization.



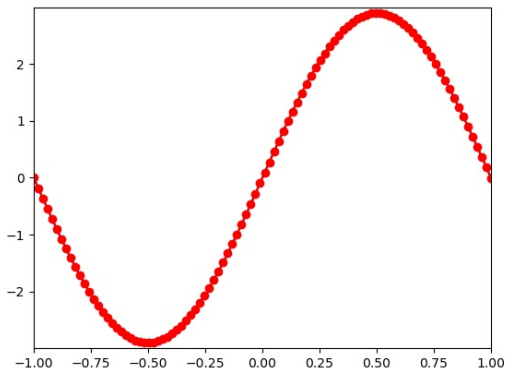
(c) $u(x)$ after 50 epochs.



(d) $u(x)$ after 50 epochs.



(e) $u(x)$ after 100 epochs.



(f) $u(x)$ after 100 epochs.

Figure 7: Neural network predictions (red) and the corresponding analytical solution (black) for random (left) and database (right) initialization after 0 (top), 50 (middle) and 100 (bottom) epochs. $\nu = 2.9$.

1. database initialization leads to better initial and final accuracy as well as faster convergence

2. the boost of transfer learning correlates with underlying similarity

4.2 Burger's Equation

4.2.1 Boundary-Initial Value Problem

The second example will be on Burger's equation: this partial differential equation is a special case of the Navier-Stokes equations [35] where the pressure gradient term is neglected. This assumption is valid for various applications, including fluid mechanics, nonlinear acoustics and gas dynamics [36]. In line with the pioneering study of Raissi et al. [5], the time-dependent one-dimensional Burger's equation with Dirichlet boundary conditions will be studied:

$$\begin{cases} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} - \nu\frac{\partial^2 u}{\partial x^2} = 0 & (x,t) \in [-1,1] \times [0,1] \\ u(t=0,x) = -sin(\pi x) & \\ u(t,x=-1) = u(t,x=1) = 0 & \end{cases} \tag{4.2}$$

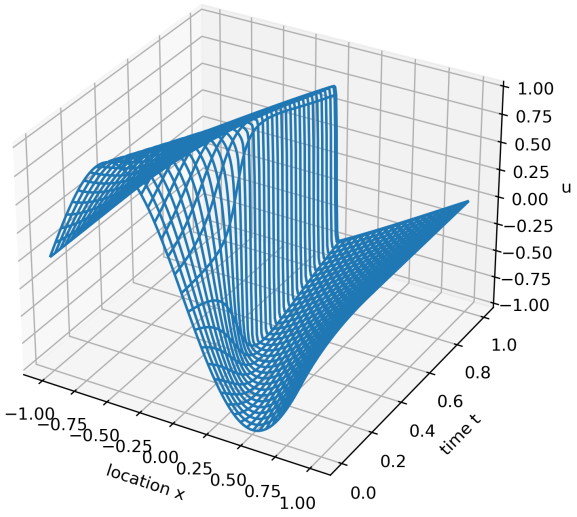
The spatio-temporal domain as well as initial and boundary conditions are fixed within the database $\mathcal{D}$. The physical parameter vector will therefore only contain the viscosity $\mathcal{P} = \begin{pmatrix} \nu \end{pmatrix}$. The viscosity parameter is critical as small values of ν can lead to shock formation with very high gradients, posing problems to conventional numerical methods.

4.2.2 Database Generation

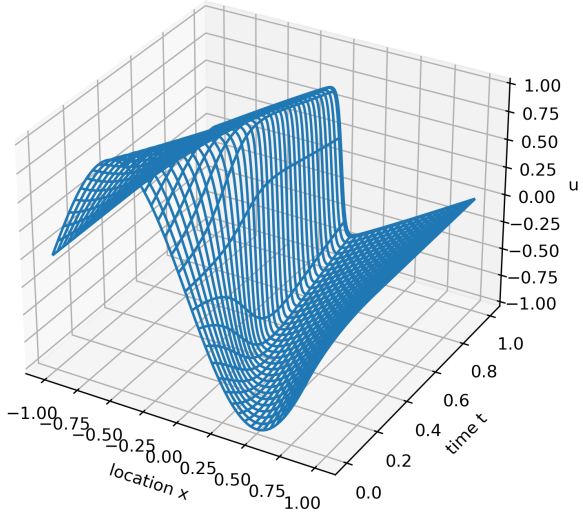
In this first step, $N^{sor} = 10$ PINNs with physical parameter vector $\mathcal{P}_i^{sor} = \begin{pmatrix} i\frac{0.01}{\pi} \end{pmatrix}, i = 1, ..., 10$, are trained. Each PINN infers the scalar quantity $\hat{\mathbf{u}} = \begin{pmatrix} \hat{u} \end{pmatrix}$ from two input neurons $\mathbf{x}^0 = \begin{pmatrix} x & t \end{pmatrix}$. A number of predicted solutions of the source PINNs are depicted in figure 8.

There is no need to do an extensive hyperparameter search, for example with regards to the best neural network architecture, as Raissi et al. [5] have studied the problem in-depth. They have performed systematic studies with respect to the number of hidden layers, hidden units, number of discrete points in the spatio-temporal domain and on temporal and spatial boundaries. With respect to an analytical reference solution, that is available for this type of problem [36], their best hyperparameter configuration reaches a relative $\mathcal{L}_2$ error of $6.7 \cdot 10^{-4}$. The underlying hyperparameters as well as their optimization strategy are summarized in table 2.

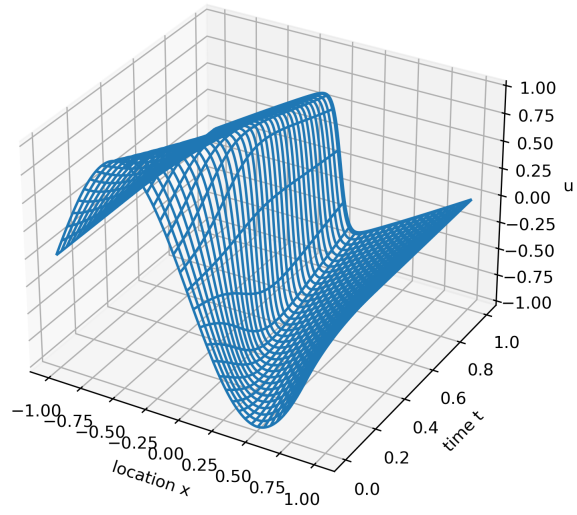
Each of the 10 physics informed neural networks was trained for approximately 112 seconds on a GeForce RTX 2080 Ti. The relative $\mathcal{L}_2$ errors for each network are reported in table 3 and illustrate that the physics-informed neural network is able to capture a close-to-exact solution in all cases. With an average error of $\mathcal{L}_2 = 6.0 \cdot 10^{-4}$, the accuracy is comparable to $\mathcal{L}_2 = 6.7 \cdot 10^{-4}$ reported in [6].



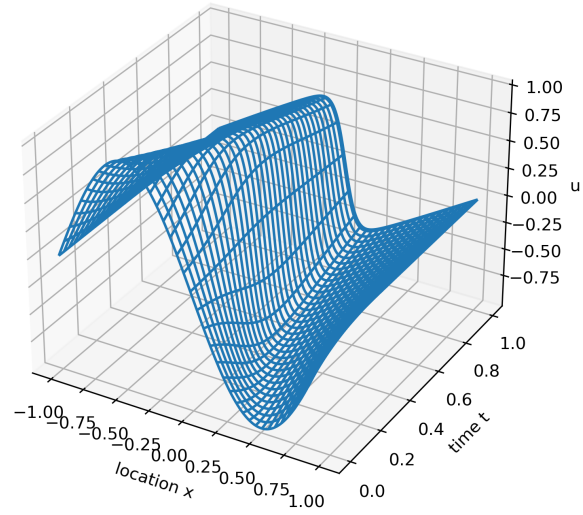
(a) $\nu = \frac{0.01}{\pi}$.



(b) $\nu = \frac{0.04}{\pi}$.



(c) $\nu = \frac{0.07}{\pi}$.



(d) $\nu = \frac{0.1}{\pi}$.

Figure 8: Neural network predictions of Burger's equation for four viscosity values ν. Low values of ν (top-left) with dominating advective term lead to sharp shock formation. Higher viscosities (top-right, bottom-left) gradually strengthen the diffusive part and lead to smoother solutions (bottom-right).

Hidden layers	8
Hidden units	20
Activation function	tanh
Domain points	10.000
Spatial/temporal boundary points	100
First optimizer	Adam, 10.000 epochs
Second optimizer	L-BFGS-B

Table 2: Hyperparameter configuration for the time-dependent one-dimensional Burger's equation.

i	$\mathcal{P}^{sor} = \left(\nu\right)$	Relative $\mathcal{L}_2$ error
1	$\frac{0.01}{\pi}$	$1.5 \cdot 10^{-3}$
2	$\frac{0.02}{\pi}$	$7.9 \cdot 10^{-4}$
3	$\frac{0.03}{\pi}$	$5.6 \cdot 10^{-4}$
4	$\frac{0.04}{\pi}$	$3.8 \cdot 10^{-4}$
5	$\frac{0.05}{\pi}$	$2.6 \cdot 10^{-4}$
6	$\frac{0.06}{\pi}$	$7.8 \cdot 10^{-4}$
7	$\frac{0.07}{\pi}$	$6.0 \cdot 10^{-4}$
8	$\frac{0.08}{\pi}$	$4.7 \cdot 10^{-4}$
9	$\frac{0.06}{\pi}$	$3.4 \cdot 10^{-4}$
10	$\frac{0.1}{\pi}$	$2.6 \cdot 10^{-4}$

Table 3: Relative $\mathcal{L}_2$ errors for each $N^{sor} = 10$ physical parameter configurations.

Optimization strategy	Average time [s]
L-BFGS (random)	52
L-BFGS (smart)	7
Adam + L-BFGS (random)	113

Table 4: Time until convergence of three different optimizers. Averaged across all 10 target models.

4.2.3 Database Initialization

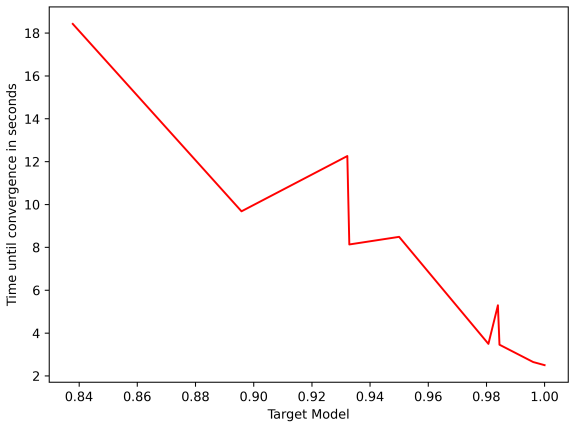
As there is access to a database of $N^{sor} = 10$ converged PINNs, a random set of new problems is generated. More specifically, $N^{tar} = 10$ random viscosity values in the range of $\mathcal{P}^{sor}$, i.e. $\nu \in [\frac{0.01}{\pi}, \frac{0.1}{\pi}]$, are constructed.

Three optimization strategies are employed and compared:

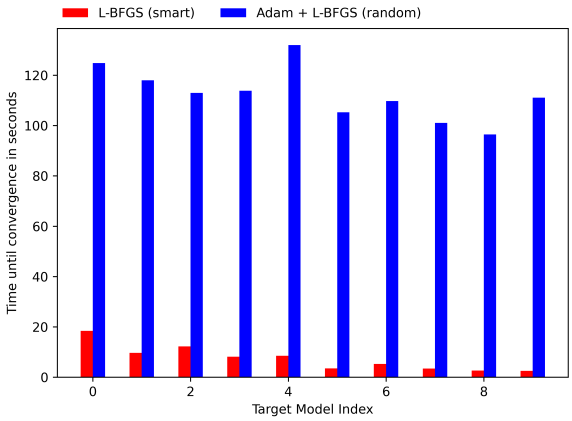
1. L-BFGS (random) with random initialization
2. L-BFGS (smart) with source initialization: approach of algorithm 1
3. Adam + L-BFGS (random) with random initialization

The choice is grounded on the following line of thoughts: strategy 1 may not reach an accuracy as good as strategy 2 but could be comparable in terms of computational cost as the optimizer is the same. In contrast, strategy 3 follows a more costly optimization, resulting in an accuracy that could become competitive to strategy 2 but likely with higher computational cost. Strategies 1 and 3 therefore each provide a means to compare either computational cost or accuracy while sacrificing the other.

Efficiency Comparison In table 4, the efficiency of algorithm 1 is evaluated: the proposed strategy is faster than both other optimization strategies. Needless to say, Adam + L-BFGS (random) employs a costly additional optimization of 10.000 epochs in advance and is therefore more time-consuming. Yet, the initialization of L-BFGS has a strong influence on the training time. With smart initialization from the database, efficiency can be significantly reduced with respect to random initialization. In addition, it seems to be the case that higher similarities result in lower computational effort (figure 9a). From a more quantitative perspective, the proposed algorithm provides an approximately 6−fold (lowest similarity) to 44−fold (highest similarity) speed-up with respect to Adam + L-BFGS (random) (figure 9b). Both observations strongly reinforce the introduction of a similarity measure $\mathcal{S}$ based on a database $\mathcal{D}$.



(a) Computational time of algorithm 1 with respect to similarity $\mathcal{S}(\nu)$.



(b) Computational time of algorithm 1 with respect to the standard optimization procedure.

Figure 9: Efficiency assessment of algorithm 1 for Burger's equation.

Optimization strategy	Average $\mathcal{L}_2$	Average $\mathcal{L}$
L-BFGS (random)	$3.6 \cdot 10^{-4}$	$1.1 \cdot 10^{-3}$
L-BFGS (smart)	$2.4 \cdot 10^{-4}$	$7.9 \cdot 10^{-4}$
Adam + L-BFGS (random)	$2.8 \cdot 10^{-4}$	$8.9 \cdot 10^{-4}$

Table 5: Comparison of three optimization strategies in terms of accuracy.

Accuracy Comparison Subsequently, all three optimization strategies are compared with respect to the final accuracy of the solution. Table 5 indicates that algorithm 1 outperforms the other two, even though all three training procedures result in very accurate solutions. From an optimization perspective, the initial guess seems to be located in a better basin and L-BFGS can converge quicker and to a better minimum. In addition, one may note that the relative error between predicted and exact solution, $\mathcal{L}_2$ correlates with the loss of a physics-informed neural network, $\mathcal{L}$.

These results suggest that transfer learning might enable to find solutions that are more accurate than with other strategies. In other words, better local minima could be identified with transfer learning while other optimization strategies get stuck in worse local minima.

4.2.4 Summary

In this section, Burger's equation has been approximated as a forward problem. We have seen that algorithm 1

1. is more efficient than both other common optimization strategies

2. becomes more efficient with increasing similarity

3. can lead a level of accuracy that is not achievable by other training procedures

4.3 Kovasznay Flow

4.3.1 Boundary Value Problem

Here, the steady two-dimensional Kovasznay flow will be studied. It is convenient to evaluate as an analytical solution for the problem is available. Yet, compared to Burger's equation, the problem is of higher dimension and comparably complex nature, hence providing an interesting task. The Kovasznay flow is described by the Navier-Stokes equations with momentum (force) balance in both x and y direction as well as the continuity equation ensuring mass conservation

$$\begin{cases} u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + \frac{\partial p}{\partial x} - \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2}\right) = 0 \quad (x,y) \in [-0.5, 1.0] \times [-0.5, 1.5] \\ u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + \frac{\partial p}{\partial y} - \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2}\right) = 0 \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} = 0 \end{cases} \tag{4.3}$$

with Reynolds number Re, horizontal velocity u, vertical velocity v and pressure p. The analytical solution is given by

$$\begin{cases} u(x,y) = 1 - e^{\lambda x} cos(2\pi y) \\ v(x,y) = \frac{\lambda}{2\pi} e^{\lambda x} sin(2\pi y) \\ p(x,y) = \frac{1}{2}(1 - e^{2\lambda x}) \end{cases} \tag{4.4}$$

with $\lambda = \frac{1}{2\nu} - \sqrt{\frac{1}{4\nu^2} + 4\pi^2}$ an viscosity $\nu = \frac{1}{Re}$. For u and v, boundary conditions are supplemented at all 4 spatial boundaries, while the outflow boundary at $x = 1.5$ is the only reference for the pressure p.

4.3.2 Database Generation

At first, a database of $N^{sor} = 5$ models is generated with physical parameter vector $\mathcal{P}_i^{sor} = \left(20i\right), i = 1, ..., 5$. Each PINN infers the three scalar quantities $\hat{\mathbf{u}} = \begin{pmatrix} \hat{u} & \hat{v} & \hat{p} \end{pmatrix}$ from two input neurons $\mathbf{x}^0 = \begin{pmatrix} x & y \end{pmatrix}$. Again, the hyperparameters are chosen with respect to another study that has performed systematic studies with respect to the network architecture [12]. Their hyperparameter configuration is adopted, while the optimization strategy is adapted: instead of costly pre-training for 110.000 epochs with the Adam optimizer, 10.000 epochs are sufficient to achieve the same level of accuracy. All hyperparameters are summarized in table 6.

Each model takes approximately 238 seconds training and the overall accuracy is better than in [12] for the velocities u and v as well as p, see table 7.

For all values of Re, the accuracy for u is better than $7.6 \cdot 10^{-4}$, for v better than $4.12 \cdot 10^{-3}$ and for p better than $5.16 \cdot 10^{-4}$ reported in [12]. This is remarkable, especially

Hidden layers	4
Hidden units	50
Activation function	tanh
Domain points	2.601
Spatial boundary points	400
First optimizer	Adam, 10.000 epochs
Second optimizer	L-BFGS-B

Table 6: Hyperparameter configuration for the steady two-dimensional Kovasznay flow.

i	$\mathcal{P}^{sor} = \left(Re\right)$	$\mathcal{L}_{2,u}$	$\mathcal{L}_{2,v}$	$\mathcal{L}_{2,p}$
1	20	$4.0 \cdot 10^{-4}$	$1.5 \cdot 10^{-3}$	$7.1 \cdot 10^{-4}$
2	40	$1.7 \cdot 10^{-4}$	$1.1 \cdot 10^{-3}$	$3.9 \cdot 10^{-4}$
3	60	$2.3 \cdot 10^{-4}$	$2.1 \cdot 10^{-3}$	$8.2 \cdot 10^{-4}$
4	80	$1.9 \cdot 10^{-4}$	$1.7 \cdot 10^{-3}$	$4.8 \cdot 10^{-4}$
5	100	$2.3 \cdot 10^{-4}$	$2.5 \cdot 10^{-3}$	$8.5 \cdot 10^{-4}$

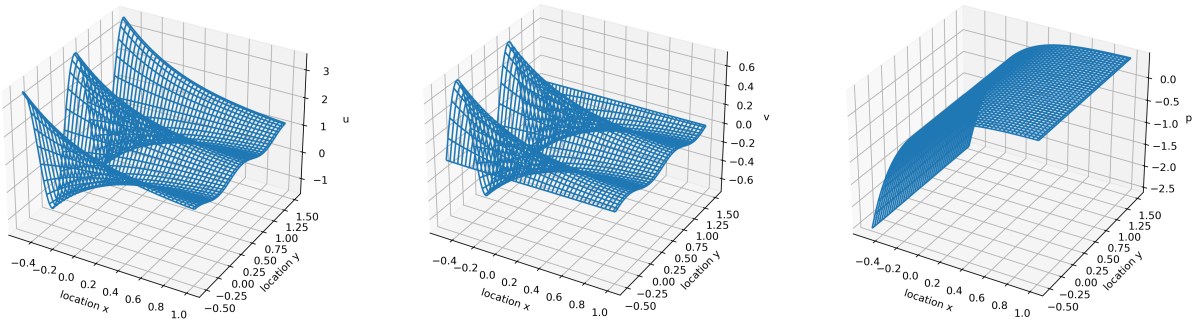
Table 7: Relative $\mathcal{L}_2$ errors for each $N^{sor} = 5$ physical parameter configurations.

for low Reynolds numbers where the variations across the spatial domain are significant (figure 10).

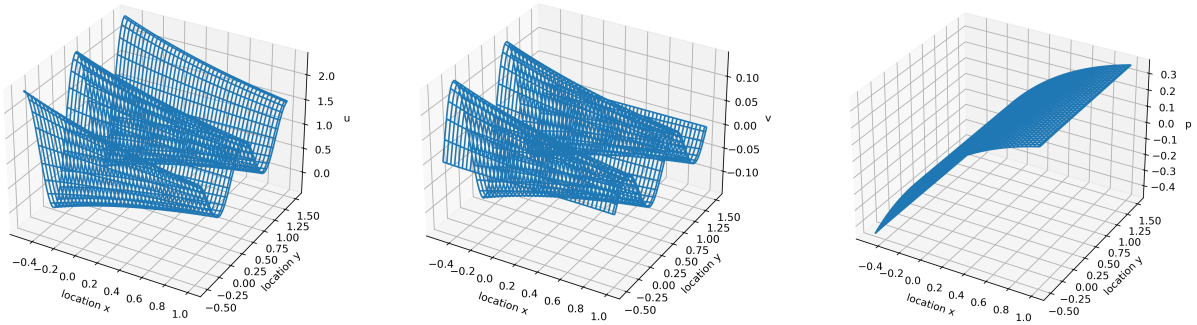
4.3.3 Database Initialization

Here, the same three optimization strategies as in the previous case for Burger's equation 4.2 are studied. $N^{tar} = 10$ new target models in the range $Re \in [20, 100]$ are randomly generated.

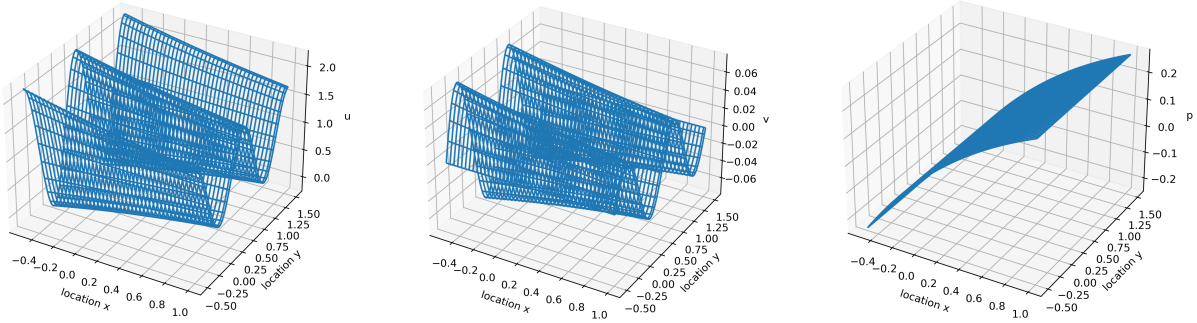
Efficiency Comparison The efficiency of all three training algorithms is compared in table 8. One can clearly see the superiority of algorithm 1 over both strategies based on random initialization. It is more efficient than its random counterpart, suggesting that a trained similar model is a better starting point for optimization. Needless to say, additional pre-training with Adam is obviously even more costly. Even more interesting, the efficiency curve of algorithm 1 in figure 11a shows a strong dependency with similarity: the closer $\mathcal{P}^{sor}$ to $\mathcal{P}^{tar}$, the more efficient and less costly the training process. From a more quantitative perspective, the proposed algorithm provides a speed-up of factor approximately 3 (lowest similarity) to 29 (highest similarity) with respect to Adam + L-BFGS (random) (figure 11b).



(a) $u(x, y)$ for $Re = 20$. (b) $v(x, y)$ for $Re = 20$. (c) $p(x, y)$ for $Re = 20$.



(d) $u(x, y)$ for $Re = 60$. (e) $v(x, y)$ for $Re = 60$. (f) $p(x, y)$ for $Re = 60$.

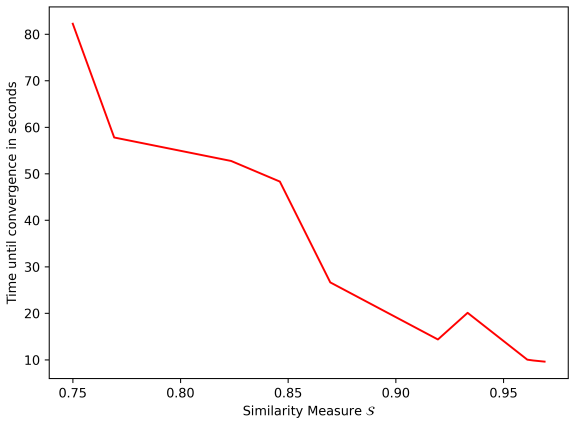


(g) $u(x, y)$ for $Re = 100$. (h) $v(x, y)$ for $Re = 100$. (i) $p(x, y)$ for $Re = 100$.

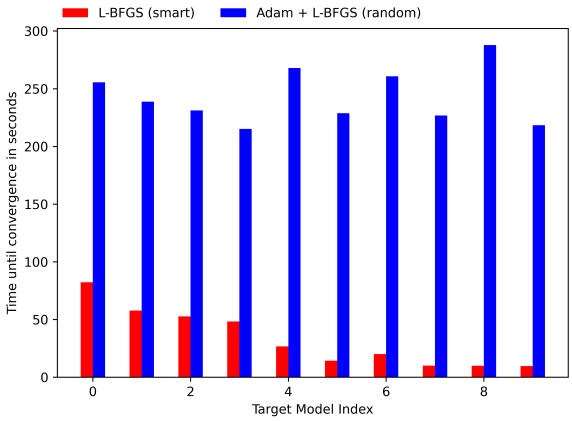
Figure 10: Neural network predictions for the steady two-dimensional Kovasznay flow. For three different Reynolds numbers, $Re = 20$ (top), $Re = 60$ (middle) and $Re = 100$ (bottom), each horizontal velocity u (left), vertical velocity v (middle) and pressure p (right) are shown.

Optimization strategy	Average time [s]
L-BFGS (random)	110
L-BFGS (smart)	33
Adam + L-BFGS (random)	243

Table 8: Time until convergence of three different optimizers. Averaged across all 10 target models.



(a) Computational time of algorithm 1 with respect to similarity $\mathcal{S}(\nu)$.



(b) Computational time of algorithm 1 with respect to the standard optimization procedure.

Figure 11: Efficiency assessment of algorithm 1 for the Kovasznay flow.

Optimization strategy	Average $\mathcal{L}_{2,u}$	Average $\mathcal{L}_{2,v}$	Average $\mathcal{L}_{2,p}$	Average $\mathcal{L}$
L-BFGS (random)	$4.6 \cdot 10^{-4}$	$3.3 \cdot 10^{-3}$	$1.5 \cdot 10^{-3}$	$5.8 \cdot 10^{-4}$
L-BFGS (smart)	$2.2 \cdot 10^{-4}$	$1.9 \cdot 10^{-3}$	$6.8 \cdot 10^{-5}$	$3.6 \cdot 10^{-4}$
Adam + L-BFGS (random)	$2.3 \cdot 10^{-4}$	$2.1 \cdot 10^{-3}$	$8.9 \cdot 10^{-4}$	$3.7 \cdot 10^{-4}$

Table 9: Comparison of three optimization strategies in terms of accuracy.

Accuracy Comparison As previously, the three training procedures are evaluated based on the final predictions' accuracy. First and foremost, algorithm 1 clearly outperforms both other optimization strategies. The relative errors to the exact analytical solution are approximately half as low as for the L-BFGS (random) strategy. Adam + L-BFGS (random) achieves better accuracy, but still not as good as L-BFGS (smart). This is the case for all three hidden variables: horizontal velocity u, vertical velocity v and pressure p. Besides, one can state again that the loss of the physics-informed neural network $\mathcal{L}$ correlates with the relative errors $\mathcal{L}_{2,i}$.

4.3.4 Summary

In this section, the Kovasznay flow has been approximated as a forward problem. We have again seen that algorithm 1

1. is more efficient than both other common optimization strategies
2. becomes more efficient with increasing similarity
3. can lead a level of accuracy that is not achievable by other training procedures

4.4 Beltrami Flow

4.4.1 Boundary-Initial Value Problem

In this final section, algorithm 1 will be investigated for more complicated flow phenomena. This should exemplify the exploitation of databases for high-dimensional and complex real-world problems. To do so, a transient flow in three-dimensional space, described by the Navier-Stokes equations will be investigated

$$\begin{cases} \frac{\partial u}{\partial t} + u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z} + \frac{\partial p}{\partial x} - \frac{1}{Re}\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right) = 0 \\ \frac{\partial v}{\partial t} + u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z} + \frac{\partial p}{\partial y} - \frac{1}{Re}\left(\frac{\partial^2 v}{\partial x^2} + \frac{\partial^2 v}{\partial y^2} + \frac{\partial^2 v}{\partial z^2}\right) = 0 \\ \frac{\partial w}{\partial t} + u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z} + \frac{\partial p}{\partial z} - \frac{1}{Re}\left(\frac{\partial^2 w}{\partial x^2} + \frac{\partial^2 w}{\partial y^2} + \frac{\partial^2 w}{\partial z^2}\right) = 0 \\ \frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z} = 0 \end{cases} \tag{4.5}$$

for the cube $(x, y, z) \in [0, 1.0] \times [0, 1.0] \times [0, 1.0]$ and the time interval $t \in [0, 1.0]$. The first three equations enforce a momentum balance in x, y, and z direction while the fourth equation ensures mass conservation. Ethier et al. [37] have found an analytical solution to this problem for a rather complex and challenging scenario. Even though this flow field is unlikely to be physically realized, it is well suited for benchmarking, i.e. to assess the accuracy of numerical solvers. The analytical solution, to which the predicted field of a PINN will be compared, is given by

$$\begin{cases} u(x,y,z) &= -a\left(e^{ax}sin(ay+dz) + e^{az}cos(ax+dy)\right)e^{-d^2t} \\ v(x,y,z) &= -a\left(e^{ay}sin(az+dx) + e^{ax}cos(ay+dz)\right)e^{-d^2t} \\ w(x,y,z) &= -a\left(e^{az}sin(ax+dy) + e^{ay}cos(az+dx)\right)e^{-d^2t} \\ p(x,y) &= -\frac{a^2}{2}\Big(e^{2ax} + e^{2ay} + e^{2az} + 2sin(ax+dy)cos(az+dx)e^{a(y+z)} \\ & \quad +2sin(ay+dz)cos(ax+dy)e^{a(z+x)} \\ & \quad +2sin(az+dx)cos(ay+dz)e^{a(x+y)}\Big)e^{-d^2t} \end{cases} \tag{4.6}$$

where a and d are free parameters to choose.

4.4.2 Database Generation

As usual, several source physics-informed neural networks need to be trained and saved in a database. Due to the computational expensiveness, only $N^{sor} = 2$ models are stored. Each PINN infers the four scalar quantities $\hat{\mathbf{u}} = \begin{pmatrix} \hat{u} & \hat{v} & \hat{w} & \hat{p} \end{pmatrix}$ from four input neurons $\mathbf{x}^0 = \begin{pmatrix} x & y & z & t \end{pmatrix}$. As for the Kovasznay flow, the publication on NSFnets [12] is used as a reference. The authors have scrutinized the problem and found a hyperparameter configuration that leads to good accuracy, so it will be used here as well (table 10).

Hidden layers	4
Hidden units	50
Activation function	tanh
Domain points	50.000
Spatial boundary points	5.000
Temporal boundary points	5.000
First optimizer	Adam, 30.000 epochs
Second optimizer	L-BFGS-B

Table 10: Hyperparameter configuration for the transient three-dimensional Beltrami flow.

i	$\mathcal{P}_{source} = \begin{pmatrix} d \end{pmatrix}$	$\mathcal{L}_{2,u}(t_0)$	$\mathcal{L}_{2,v}(t_0)$	$\mathcal{L}_{2,w}(t_0)$	$\mathcal{L}_{2,u}(t_1)$	$\mathcal{L}_{2,v}(t_1)$	$\mathcal{L}_{2,w}(t_1)$
1	0.75	$2.9 \cdot 10^{-4}$	$3.3 \cdot 10^{-4}$	$2.9 \cdot 10^{-4}$	$6.0 \cdot 10^{-4}$	$7.1 \cdot 10^{-4}$	$1.1 \cdot 10^{-3}$
2	1.00	$4.0 \cdot 10^{-4}$	$3.5 \cdot 10^{-4}$	$3.9 \cdot 10^{-4}$	$1.3 \cdot 10^{-3}$	$1.2 \cdot 10^{-3}$	$1.9 \cdot 10^{-3}$

Table 11: Relative $\mathcal{L}_2$ errors for each of the $N^{sor} = 2$ physical parameter configurations at initial time $t_0 = 0$ and final time $t_1 = 1$.

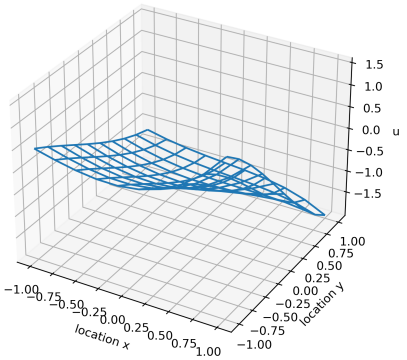
It is important to note that the above analytical solution in (4.6) is very sensitive with respect to a and d. Stated differently, small variations in a and d can lead to entirely different solutions $\begin{pmatrix} u & v & w & p \end{pmatrix}$. To construct a database $\mathcal{D}$ that covers a similar breadth of solutions as in the previous chapters, the considered variations are comparably small with $a = 1$ fixed and $d \in [0.75, 1]$, so $\mathcal{P} = \begin{pmatrix} d \end{pmatrix}$. In fact this variation in parameters a and d results in solutions (u, v, w, p) of which the variation is as big as in chapter 4.3, based on the analytical solutions.

The converged models are evaluated based on their accuracy at two time instances: $t_0 = 0$, i. e. satisfaction of initial conditions and $t_1 = 1$, at the end. The relative $\mathcal{L}_2$ error is small in all cases (table 11) and in a similar range than reported in the reference study [12]. The errors at t_1 are significantly higher than at t_0, a pattern also observed in [12].

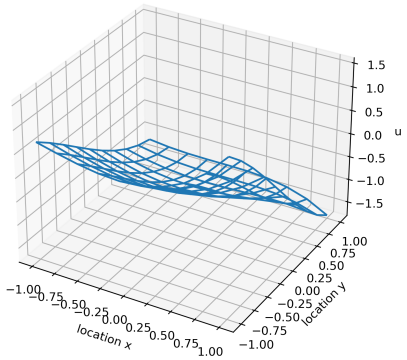
Training for each model took close to 1.5 hours with 5.100 seconds. To get a better impression of the Beltrami flow, several neural network's predictions are plotted in figure 12.

4.4.3 Database Initialization

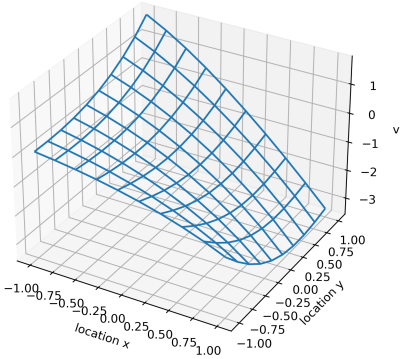
Here, $N^{tar} = 5$ new target models with $d \in [0.75, 1]$ are generated and each model is trained on the same three optimization strategies as in the previous examples.



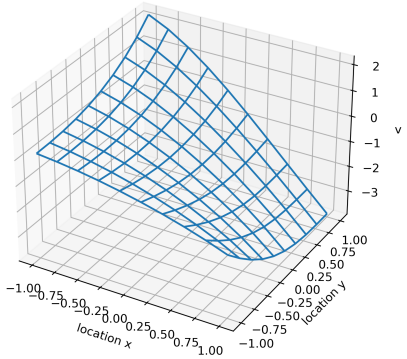
(a) $u(x, y, z = 0, t = 0)$ for $d = 0.75$.



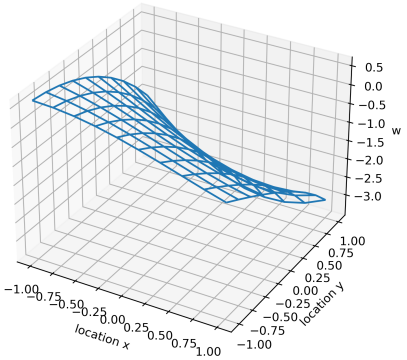
(b) $u(x, y, z = 0, t = 0)$ for $d = 1$.



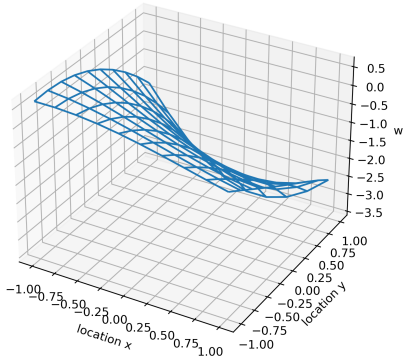
(c) $v(x, y, z = 0, t = 0)$ for $d = 0.75$.



(d) $v(x, y, z = 0, t = 0)$ for $d = 1$.



(e) $w(x, y, z = 0, t = 0)$ for $d = 0.75$.

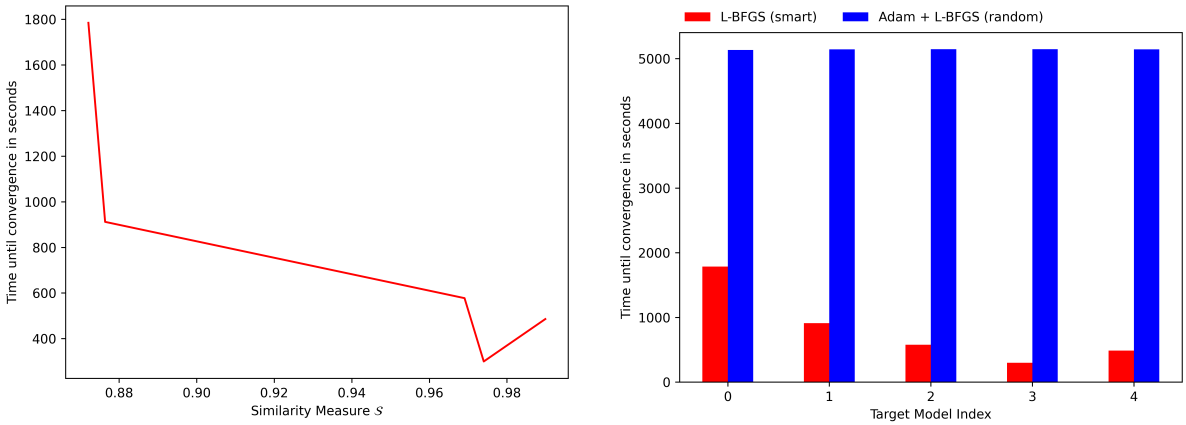


(f) $w(x, y, z = 0, t = 0)$ for $d = 1$.

Figure 12: Neural network predictions for the transient three-dimensional Beltrami flow. For two different values of $d = 0.75$ (left) and $d = 1$ (right), each horizontal velocity u (top), lateral velocity v (middle) and vertical velocity w (bottom) are shown at the slice $z = 0$ and time instance $t = 0$.

Optimization strategy	Average time [s]
L-BFGS (random)	1786
L-BFGS (smart)	812
Adam + L-BFGS (random)	5143

Table 12: Time until convergence of three different optimizers. Averaged across all 5 target models.



(a) Computational time of algorithm 1 with respect to similarity $\mathcal{S}(\nu)$.

(b) Computational time of algorithm 1 with respect to the standard optimization procedure.

Figure 13: Efficiency assessment of algorithm 1 for the Beltrami flow.

Efficiency Comparison Table 12 clearly indicates the superiority of algorithm 1 over both other optimization strategies. It is reasonable that pre-training with Adam optimizer induces a more costly optimization process. More interestingly, the initialization has a strong effect on the convergence time of the L-BFGS optimizer, as database initialization results in a more than twofold speed up with respect to random initialization.

Furthermore, one can observe a correlation between time until convergence and similarity $\mathcal{S}(d)$: as similarity increases the overall duration of optimization decreases (figure 13a). This resulted in speed-ups of 3 (lowest similarity) up to 17 (second-highest similarity) (figure 13b).

Accuracy Comparison In addition to the efficiency improvement, transfer learning boosted the training in terms of accuracy: algorithm 1 outperforms both other optimization strategies consistently. It is particularly remarkable that higher levels of accuracy can be achieved than with the standard training procedure of Adam + L-BFGS (random) (see table 13). Therefore, database initialization is so effective that the optimization procedure ends up in local minima that are better than anything possible with random

Optimization strategy	$\mathcal{L}_{2,u}$	$\mathcal{L}_{2,v}$	$\mathcal{L}_{2,w}$
L-BFGS (random)	$6.4 \cdot 10^{-4}$	$6.3 \cdot 10^{-4}$	$5.9 \cdot 10^{-4}$
L-BFGS (smart)	$3.3 \cdot 10^{-4}$	$3.4 \cdot 10^{-4}$	$3.3 \cdot 10^{-4}$
Adam + L-BFGS (random)	$3.7 \cdot 10^{-4}$	$3.6 \cdot 10^{-4}$	$3.6 \cdot 10^{-4}$

(a) $t = 0$

Optimization strategy	$\mathcal{L}_{2,u}$	$\mathcal{L}_{2,v}$	$\mathcal{L}_{2,w}$
L-BFGS (random)	$1.8 \cdot 10^{-3}$	$1.9 \cdot 10^{-3}$	$2.0 \cdot 10^{-3}$
L-BFGS (smart)	$8.5 \cdot 10^{-4}$	$8.7 \cdot 10^{-4}$	$1.3 \cdot 10^{-3}$
Adam + L-BFGS (random)	$1.3 \cdot 10^{-3}$	$1.3 \cdot 10^{-3}$	$1.4 \cdot 10^{-3}$

(b) $t = 1$

Table 13: Accuracy measured in the relative $\mathcal{L}_2$ norm with respect to the analytical solution. The average are computed across all 5 target models and at two time instances: $t_0 = 0$ (top) and $t_1 = 1$ (bottom).

initialization. The accuracy decreases from t_0 to t_1 which is again consistent with the results from the section on database generation (see chapter 4.4.2).

4.4.4 Summary

In this section, the Beltrami flow has been approximated as a forward problem. We have again seen that algorithm 1

1. is more efficient than both other common optimization strategies
2. becomes more efficient with increasing similarity
3. can lead a level of accuracy that is not achievable by other training procedures
4. is applicable for more complex problem statements (BIVPs)

5 Conclusion

5.1 Summary

This bachelor's thesis has introduced a new paradigm in training physics-informed neural networks. Instead of random initialization, applied in almost all previous works in scientific machine learning, databases with converged physics-informed neural networks are exploited. After matching a new target problem of interest with its most similar problem from a database, the trained network parameters of that source model are taken as an initialization for the target model. Therefore, a principled framework for initialization has been presented.

Across a wide range of examples, the proposed framework has been carefully analyzed with respect to common transfer learning metrics. The application on the Poisson equation has given an intuitive understanding of transfer learning in the scope of physics-informed neural networks. As the initial parameters lead to an educated guess, the initial approximation has been much better than if random initialization were employed. During training, both the speed of convergence and the final accuracy were superior.

Subsequently, more sophisticated examples of fluid dynamics with Burger's equation, the Kovasznay and Beltrami flow have served as applications for the presented database approach. Across all examples, the proposed algorithm has outperformed other optimization strategies in terms of speed of convergence, up to almost two orders of magnitude. The acceleration through a smart initialization correlated with the underlying similarity of two problems. This observation reinforces the property of such a database being self-reinforcing: denser databases present a larger likelihood of containing a trained model of which the physical problem setup is similar to the target model. Therefore, the effectiveness of the database increases with its number of entries, i.e. the number of trained and stored physics-informed neural networks. Even more surprisingly, the proposed algorithm has yielded a final accuracy that has always been superior to other optimization strategies.

5.2 Limitations and Future Work

The physical parameter vector, describing the boundary-initial value problem, has been one-dimensional in all examples. Yet, the framework has been created with the idea that $\mathcal{P}$ can be multi-dimensional: several entries in the physical parameter vector would allow to match more diverse problems for initialization. The meaning of entries additional to a characteristic number may include a parametrized geometry, discretized initial and boundary conditions or possibly material parameters. Needless to say, the introduced

similarity measure $\mathcal{S}$ is applicable to higher-dimensional problem descriptions.

Another possible use case for multi-dimensional $\mathcal{P}$ is given by inverse problems: starting from observations (measurements) of the physical quantity of interest, the goal of an inverse problem is to determine the parameters that best describe the given data [7]. There, $\mathcal{P}$ would also include measurement data at specific locations and time instances. An interesting research task would be to investigate *how many* measurement points $\mathcal{P}$ should contain. Above a certain number, the proposed approach may fail due to the curse of dimensionality. For small numbers of observations, it is expected that the database approach works for inverse problems as well as for forward problems studied throughout this bachelor's thesis.

On a broader scope, smart initialization based on a database is a general idea, applicable to much more than fluid dynamics and physics-informed neural networks. The concept of describing some task by a discrete vector and mapping it to other tasks has intentionally been presented as a widely applicable framework. More specifically, problems with computationally expensive training, such as neural networks applied to image recognition and natural language processing, could benefit from database initialization and transfer learning.

Acknowledgment

I want to sincerely thank Xuhui Meng from Brown University for the supervision of this bachelor's thesis. His excellent recommendations and ideas have been very valuable. I am also grateful for the support of Lu Lu who created the library DeepXDE and continuously provided support to fix implementation issues. On top, I would like to thank George Em Karniadakis for providing me the chance to write my bachelor thesis at his group at Brown University.

List of Figures

List of Tables

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012), pp. 1097–1105.

[2] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. "Sequence to sequence learning with neural networks". In: *arXiv preprint arXiv:1409.3215* (2014).

[3] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. "Deep learning". In: *nature* 521.7553 (2015), pp. 436–444.

[4] Nathan Baker et al. *Workshop report on basic research needs for scientific machine learning: Core technologies for artificial intelligence*. Tech. rep. USDOE Office of Science (SC), Washington, DC (United States), 2019.

[5] Maziar Raissi, Paris Perdikaris, and George E Karniadakis. "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations". In: *Journal of Computational Physics* 378 (2019), pp. 686–707.

[6] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part I): Data-driven Solutions of Nonlinear Partial Differential Equations". In: *arXiv preprint arXiv:1711.10561* (2017).

[7] Maziar Raissi, Paris Perdikaris, and George Em Karniadakis. "Physics Informed Deep Learning (Part II): Data-driven Discovery of Nonlinear Partial Differential Equations". In: *arXiv preprint arXiv:1711.10566* (2017).

[8] Lu Lu et al. "DeepXDE: A deep learning library for solving differential equations". In: *arXiv preprint arXiv:1907.04502* (2019).

[9] Isaac E Lagaris, Aristidis Likas, and Dimitrios I Fotiadis. "Artificial neural networks for solving ordinary and partial differential equations". In: *IEEE transactions on neural networks* 9.5 (1998), pp. 987–1000.

[10] Georgios Kissas et al. "Machine learning in cardiovascular flows modeling: Predicting arterial blood pressure from non-invasive 4D flow MRI data using physics-informed neural networks". In: *Computer Methods in Applied Mechanics and Engineering* 358 (2020), p. 112623.

[11] Luning Sun et al. "Surrogate modeling for fluid flows based on physics-constrained deep learning without simulation data". In: *Computer Methods in Applied Mechanics and Engineering* 361 (2020), p. 112732.

[12] Xiaowei Jin et al. “NSFnets (Navier-Stokes Flow nets): Physics-informed neural networks for the incompressible Navier-Stokes equations”. In: *Journal of Computational Physics* 426 (2020), p. 109951.

[13] Ehsan Haghighat et al. “A deep learning framework for solution and discovery in solid mechanics: linear elasticity”. In: *arXiv preprint arXiv:2003.02751* (2020).

[14] Somdatta Goswami et al. “Transfer learning enhanced physics informed neural network for phase-field modeling of fracture”. In: *Theoretical and Applied Fracture Mechanics* 106 (2020), p. 102447.

[15] Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. “Adaptive activation functions accelerate convergence in deep and physics-informed neural networks”. In: *Journal of Computational Physics* 404 (2020), p. 109136.

[16] Ameya D Jagtap, Kenji Kawaguchi, and George Em Karniadakis. “Locally adaptive activation functions with slope recovery for deep and physics-informed neural networks”. In: *Proceedings of the Royal Society A* 476.2239 (2020), p. 20200334.

[17] Sifan Wang, Yujun Teng, and Paris Perdikaris. “Understanding and mitigating gradient pathologies in physics-informed neural networks”. In: *arXiv preprint arXiv:2001.04536* (2020).

[18] Remco van der Meer, Cornelis Oosterlee, and Anastasia Borovykh. “Optimally weighted loss functions for solving PDEs with Neural Networks”. In: *arXiv preprint arXiv:2002.06269* (2020).

[19] Xavier Glorot and Yoshua Bengio. “Understanding the difficulty of training deep feedforward neural networks”. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. JMLR Workshop and Conference Proceedings. 2010, pp. 249–256.

[20] Maria Catherine Borres. *Differential Equations : Theory and Applications.* Arcler Press, 2019. ISBN: 9781773615967. URL: http://search.ebscohost.com.eaccess.ub.tum.de/login.aspx?direct=true&db=nlebk&AN=2013905&site=ehost-live.

[21] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. “Multilayer feedforward networks are universal approximators”. In: *Neural networks* 2.5 (1989), pp. 359–366.

[22] Qi Zhang, Yilin Chen, and Ziyi Yang. “Data-driven solutions and discoveries in mechanics using physics informed neural network”. In: (2020).

[23] Qin Lou, Xuhui Meng, and George Em Karniadakis. "Physics-informed neural networks for solving forward and inverse flow problems via the Boltzmann-BGK formulation". In: *arXiv preprint arXiv:2010.09147* (2020).

[24] Maziar Raissi, Alireza Yazdani, and George Em Karniadakis. "Hidden fluid mechanics: Learning velocity and pressure fields from flow visualizations". In: *Science* 367.6481 (2020), pp. 1026–1030.

[25] Jens Berg and Kaj Nyström. "A unified deep artificial neural network approach to partial differential equations in complex geometries". In: *Neurocomputing* 317 (2018), pp. 28–41.

[26] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[27] Dong C Liu and Jorge Nocedal. "On the limited memory BFGS method for large scale optimization". In: *Mathematical programming* 45.1 (1989), pp. 503–528.

[28] Martin Abadi et al. "Tensorflow: A system for large-scale machine learning". In: *12th {USENIX} symposium on operating systems design and implementation ({OSDI} 16)*. 2016, pp. 265–283.

[29] Adam Paszke et al. "Automatic differentiation in pytorch". In: (2017).

[30] Atilim Gunes Baydin et al. "Automatic differentiation in machine learning: a survey". In: *Journal of machine learning research* 18 (2018).

[31] Lisa Torrey and Jude Shavlik. "Transfer learning". In: *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[32] Bruce Roy Munson et al. *Fluid mechanics*. Wiley Singapore, 2013.

[33] Prabhu Lal Bhatnagar, Eugene P Gross, and Max Krook. "A model for collision processes in gases. I. Small amplitude processes in charged and neutral one-component systems". In: *Physical review* 94.3 (1954), p. 511.

[34] LIG Kovasznay. "Laminar flow behind a two-dimensional grid". In: *Mathematical Proceedings of the Cambridge Philosophical Society*. Vol. 44. 1. Cambridge University Press. 1948, pp. 58–62.

[35] Peter Constantin and Ciprian Foias. *Navier-stokes equations*. University of Chicago Press, 1988.

[36] Cea Basdevant et al. "Spectral and finite difference solutions of the Burgers equation". In: *Computers & fluids* 14.1 (1986), pp. 23–41.

[37] C Ross Ethier and DA Steinman. “Exact fully 3D Navier–Stokes solutions for benchmarking”. In: *International Journal for Numerical Methods in Fluids* 19.5 (1994), pp. 369–375.

A Appendix

A.1 Frequently Used Variables

$\begin{pmatrix} x & y & z \end{pmatrix}$	vector with spatial coordinates x, y and z
t	time
$\begin{pmatrix} u & v & w \end{pmatrix}$	vector with velocities u, v and w
p	pressure
$\mathcal{N}(u, x, y, z, t)$	non-linear operator in a differential equation
$\mathcal{F}(x, y, z, t)$	source term in a differential equation
$\mathcal{P}$	physical parameter vector summarizing the problem statement
$\mathcal{W}$	parameter vector of a neural network with weights and biases
$\mathcal{D}$	dataset *or* database of PINNs, depending on the context
N	amount/number of a certain quantity, depending on the context

Table 14: Overview of frequently used variables with symbols (left) and explanation (right).

A.2 Subscripts and Superscripts

- a hat above a variable, e.g. $\hat{u}$ denotes the prediction of a neural network for that variable
- the superscript *sor*, e.g. $\mathcal{P}^{sor}$, is an abbreviation for source and refers to a source model that is trained from a random state and added to a database
- the superscript *tar*, e.g. $\mathcal{P}^{tar}$, is an abbreviation for target and refers to a target model that is initialized from a model in the database
- the superscript *Xav*, e.g. $\mathcal{W}^{Xav}$, is an abbreviation for Xavier and refers to Xavier initialization