PHYSICS-INFORMED NEURAL NETWORKS TO SOLVE BURGER'S EQUATION

Rune Højlund[a] (s173965), Øyvind Winton[b] (s160602)

[a]DTU Physics
[b]Niels Bohr Institute, University of Copenhagen

ABSTRACT

Physics-informed neural networks (PINNs) combine data and mathematical models in a framework for inference and system identification. In this project, we implemented a PINN to solve the non-linear Burger's equation from fluid dynamics and compared the results with a finite element method (FEM) solution. We found that the PINN was able to resemble the FEM solution using only the physics of the problem for training, with a mean squared error of $2.40 \cdot 10^{-4}$. The main discrepancies were located around the shock-wave of the solution function. We further investigated how PINNs can be used for system identification on the same equation. Finally we demonstrated the flexibility of the framework by applying it to solve an equation from glaciology with only minor modifications.

Code availability: Code for production of results and figures is available on `https://github.com/runehoejlund/deep-learning-pinn`

1. INTRODUCTION

PINNs are neural networks which are designed to honour constraints given by physical laws or other domain knowledge. This enables more accurate forward-modelling (prediction), compared to a purely data-driven approach, and reduces the amount of data needed for training. Further, it allows for inverse modeling (estimating model parameters). In this project, we investigate how PINNs are set up, trained and tweaked to improve predictive performance, using it for both prediction without data and parameter estimation. In section 2 we introduce the fundamentals of PINNs. In Section 3 we introduce the Burger's equation, which is a nonlinear partial differential equation (PDE) from fluid dynamics and will be the case study of our implementation of PINNs. We further take the reader through our optimisation of the network hyperparameters, architecture and other reflections, detailing our various attempts and their merits. In Section 4 we present our results with the tuned network. In Section 5 we show how PINNs can be used for system identification. In Section 6 we investigate the flexibility of the forward-modelling method, by applying it to a different equation from the field of glaciology with only minor modifications to the network. In Section 7 we discuss how the model could be improved.

2. PHYSICS-INFORMED NEURAL NETWORKS

Neural networks (NNs) are a branch of machine learning, where input is mapped output through a complex network of simple computations. The structure of the network resembles that of signals traveling in the brain, from which neural networks gets its name. Each neuron in the hidden layer receives a weighted sum from the input neurons, which is then parsed through a non-linear activation function. The output of each neuron is then used as an input to neurons in the following layer, and this approach is repeated for all hidden layers until the output layer is reached, where there is no activation function. This is called the **forward pass**.

The output is typically compared to observations, and a loss is calculated. The network is typically trained by gradient descent, using the derivative of the loss function w.r.t. each of the weights in the network. The derivative calculations are implemented using automatic differentiation (AD), which is an efficient method to evaluate derivatives in a computational graph using the chain rule with clever bookkeeping. All the weights are updated in an attempt to minimize the loss function. This is called **backpropagation**. Thus, the loss function defines the problem.

PINNs are an extension of neural networks which take advantage of AD being implemented in deep learning frameworks. In physics and mathematical modelling it is often possible to describe the dynamics of a system through partial differential equations (PDEs) with sufficient boundary conditions (BCs) and initial conditions (ICs). The general form of problems considered in this project is:

$$\mathcal{N}_{\text{PDE}}\{u;\nu\} = f(u, u_x, u_{xx};\nu) - u_t = 0 \tag{1a}$$

$$\mathcal{N}_{\text{BC}}\{u\} = \mathcal{B}\{u\} - g(t) = 0 \tag{1b}$$

$$\mathcal{N}_{\text{IC}}\{u\} = u(x, t=0) - h(x) = 0 \tag{1c}$$

where $u = u(x,t)$ is some function of space $x \in [x_{\min}, x_{\max}]$ and time $t \in [0, T]$, $\mathcal{N}...$ are (potentially non-linear differential) operators, $\nu \in \mathbb{R}$ is a model parameter, $\mathcal{B}$ is an operator defining the boundary conditions and subscript denotes partial differentiation, e.g. $u_t = \frac{\partial}{\partial t} u(x,t)$.

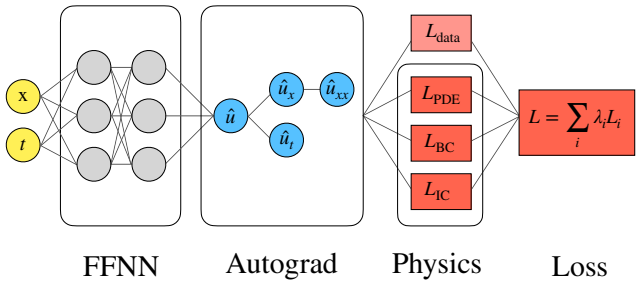


Fig. 1: Schematic of PINN architecture.

Figure 1 shows the general architecture in a PINN. The core idea is to use the output of the network $\hat{u}$ as an approximation of the true u and then apply AD to estimate how well the PDE, BCs and ICs are satisfied. We have chosen to use the mean squared error loss (MSE) without further analysis of other options. The loss function is then in general defined as

$$L = L_{\text{data}} + \underbrace{L_{\text{PDE}} + L_{\text{BC}} + L_{\text{IC}}}_{\text{physics}}, \tag{2}$$

where the physics losses are the MSEs of (1a), (1b) and (1c) with u replaced by its estimator $\hat{u}$. E.g.

$$L_{\text{PDE}} = \frac{1}{N_{\text{PDE}}} \sum_{i=1}^{N_{\text{PDE}}} \left(\mathcal{N}_{\text{PDE}}\{\hat{u}(x_i, t_i); \nu\}\right)^2,$$

with N_{PDE} beeing the number of collocation points. PINNs extends the capabilities of neural networks in multiple ways. Two key points are briefly expanded upon below.

2.1. Reduced data requirements for forward modelling

The inclusion of a physical model in the loss function, restricts the output of the network to solutions that respect the physics of the problem thus acting as regularization procedure. This reduces the amount of data necessary to find a given solution, which is a major benefit for problems where observations are sparse.

Provided that the problem is restricted by sufficient BCs and/or ICs the PINN can be trained as a numerical forward model, that takes no data to train or evaluate. It is noted that the PDE loss can typically be evaluated continuously within the domain.

2.2. System identification

PINNs can be used to identify parameters of the physical model, given that observations of $u(x, t)$ are available. This allows the usage of PINNs for parameter estimation, where a physical model and (noisy) data is available.

3. BURGER'S EQUATION

We investigate PINN's by applying it to the non-linear Burger's equation from fluid dynamics. Burger's equation with Dirichlet BCs and a sinusoidal IC is [1]

$$u_t + uu_x - \nu u_{xx} = 0, \quad x \in [-1, 1], \quad t \in [0, 1], \tag{3a}$$
$$u(t, -1) = u(t, 1) = 0. \tag{3b}$$
$$u(0, x) + \sin(\pi x) = 0, \tag{3c}$$

The equation is used in many numerical studies as a benchmark problem, posing multiple challenges due to its non-linearity, and having a non-linear convective term and a diffusive term. In this project, a finite element solution (FEM) of (3) written in Matlab is used as test-data for external validation.

3.1. Training, validation and test in relation to PINNs

For PINNs without data in the loss function, we essentially perform unsupervised learning. While literature about the field has treated this sparingly, we here motivate and define our distinction of training, validation and test data.

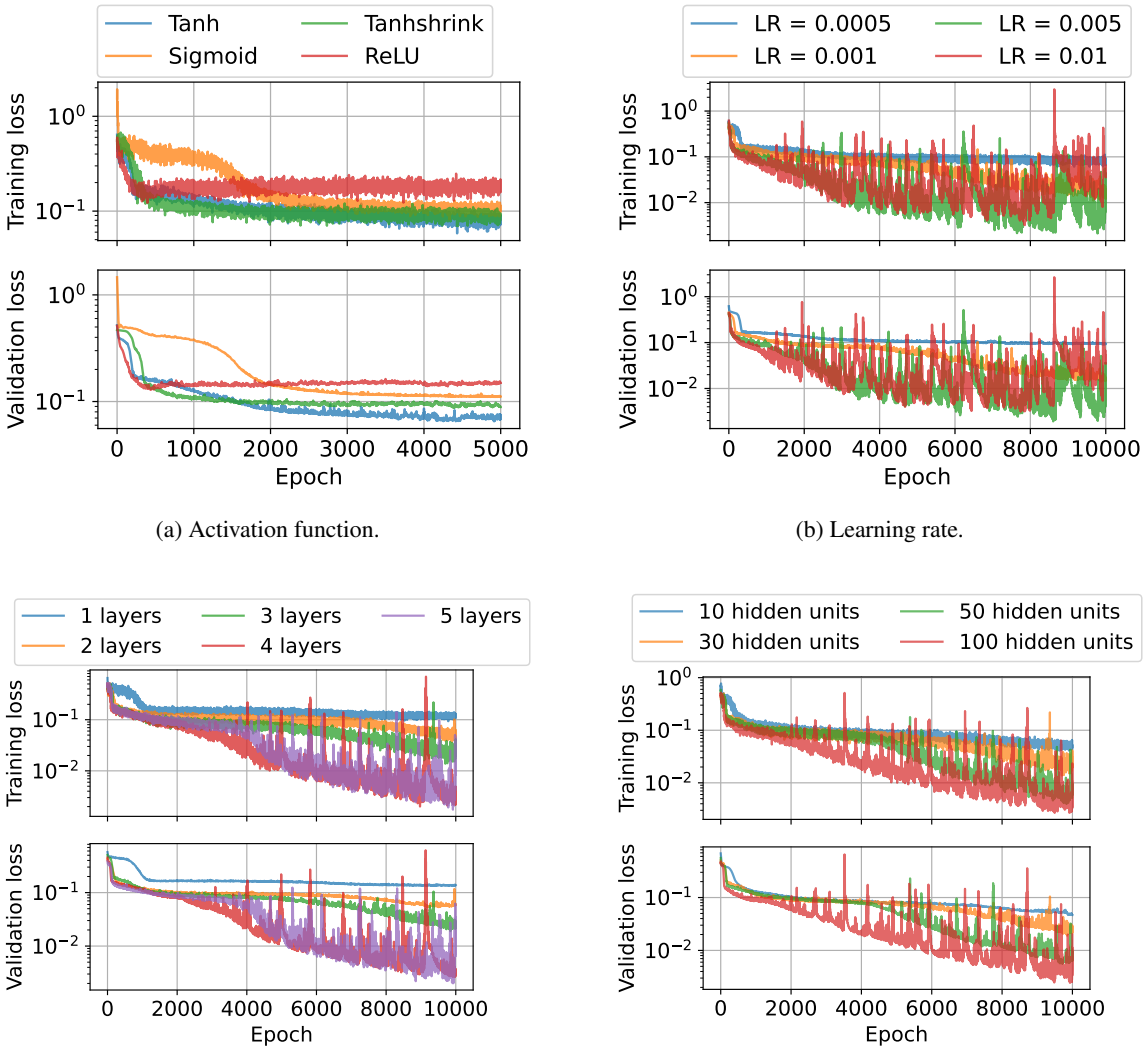
Training loss. We define the training loss as the loss function evaluated at the collocation points used in training. We define these points in different manners, as detailed later in this section.

Validation loss. We define the validation loss as the loss function evaluated on a set of collocation points not used in training. We evaluate both inner and boundary/initial collocation points sampled from uniform distributions. For a given analysis figure, the validation points remain the same.

Test loss. We define the test loss as the data MSE between our PINN prediction and the FEM solution. The FEM solution was calculated on a uniform grid spanning 301 spatial and 101 temporal points.

3.2. Network and training architecture

We implement PINN using the PyTorch package in Python. We attempted to find the optimal network architecture by sampling different hyper parameters (activation function, learning rate, number of layers and number of hidden units per layer) and evaluating the corresponding validation loss convergence. Here, it should be noted that the stochasticity of weight initialisation, collocation points and optimisation algorithm may have affected the analysis and conclusions drawn. On Figure 2 the training and validation losses are plotted for different architectures. In all cases it is seen that the training and validation loss are highly correlated, which is due to the random initialisation of collocation points as described in more detail in section 3.4. On Figure 2a, we see that Tanh closely folowed by Tanhshrink was clearly the optimal activation function. From Figure 2c and 2d it is seen that the model needs a certain capacity (4 layers with 30 units



(a) Activation function.

(b) Learning rate.

(c) Number of layers.

(d) Number of hidden units per layer.

Fig. 2: Comparison of loss convergence for different hyper parameters. The baseline model used tanh as activation function, 0.001 as learning rate, 3 layers with 30 hidden units in each and random sampling of collocation points (see section 3.4).

or 3 layers with 100 units) to solve the PDE-problem satisfyingly. However increasing the capacity to 5 layers prolonged training time without significant improvements. For the high-capacity models the PINN had not converged yet after 10 000 epochs, but the validation loss reached was still superior to lower capacity models.

3.3. Multi-objective loss scaling

It might be an issue if the different terms in the loss function (2) have different scalings such that one of the losses dominates. In an attempt to address this issue we introduced a scaling parameter λ to the loss:

$$L = \sum_i \lambda_i L_i. \tag{4}$$

The λ's were dynamically updated during optimisation. For this purpose we used the SoftAdapt algorithm ([2, 3]):

$$\lambda_i = \frac{\exp(f_i)}{\sum_j \exp(f_j)}, \quad f_i = \frac{L_i^k}{L_i^{k-1}}, \tag{5}$$

where k is the current iteration number. This approach directs optimization focus to L_j if f_j is relatively high compared to the other f_i'. In other words, the part of the loss function with the smallest decrease in the last iteration, gets more weight in the next iteration. The desired effect is to converge faster in training. We have experimented with variants of the implementation of SoftAdapt as described in [2], but found no improvements in our results, and thus discard the implementation for this project.

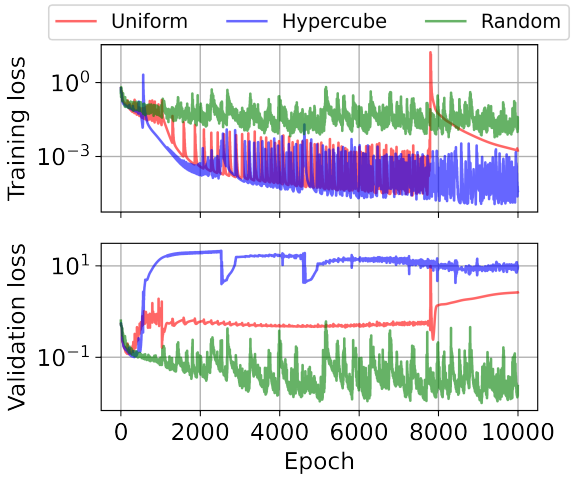


Fig. 3: Physics and data losses for three different collocation point approaches. Uniform refers to a uniform grid of collocation points. Hypercube refers to Latin hypercube sampling of points. Random refers to random sampling of interior and boundary points in each epoch.

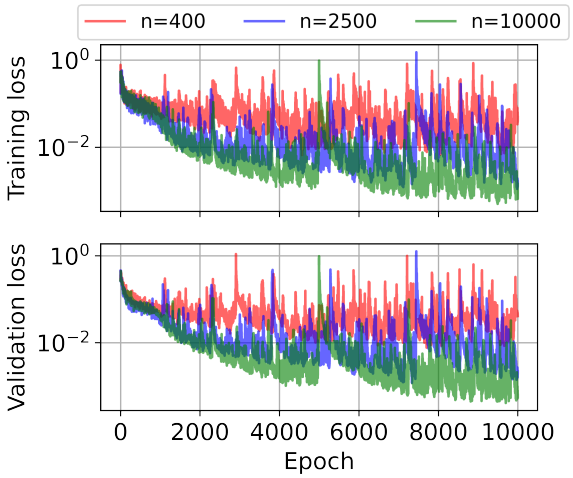


Fig. 4: Training and validation losses for increasing number of interior points using the random method described earlier.

3.4. Collocation points

Compared with traditional data-driven ML-algorithms a PINN has the advantage that the loss function can be evaluated at an unlimited number of points within the domain bounds. We have experimented with different approaches to the choice of collocation points, and present here a comparison between three different approaches: **uniform** grid (as in [3]), Latin **hypercube** sampling (LHS) (as in [4]. See also [5]) and **random** sampling from a uniform distribution, with new points drawn in each epoch. The idea with the random approach is to take advantage of the fact that the loss function is a continuous function. This will help preventing overfitting, as different collocation points are trained on in each epoch.

In Figure 3 we show the training and validation losses for each of the above approaches. We find that the uniform approach results in the lowest training errors, while the random approach has the highest training error. This reflects the fact that the uniform approach repeatedly trains on the same points, where the random approach trains on new points in each epoch. The benefit of the random approach is shown in the validation loss, where it shows the lowest losses.

The results show that while the uniform approach solves the PDE very well on the training points, it does not capture the nature of the PDE well on other points, whereas the random approach generalizes better to the whole domain. Thus, the uniform approach represents a classic example of overfitting.

With the lowest errors achieved with the random points, we further investigate how the number of collocation points used affect the training ability. The results are shown in Figure 4, where we find that more collocation points converges to a lower validation loss, albeit at a higher computational cost.

4. FORWARD MODELLING RESULTS

Following the analysis in the previous section, we train a PINN with the optimal hyperparameters and denote this the full-capacity network. The validation loss is shown in Figure 5. We find that the validation loss is substantially higher than in previous analyses, which we will discuss further in Section 7. For this reason we simplify the network architecture, reducing the number of hidden units. The choices are shown in Table 1 which we denote the final network.

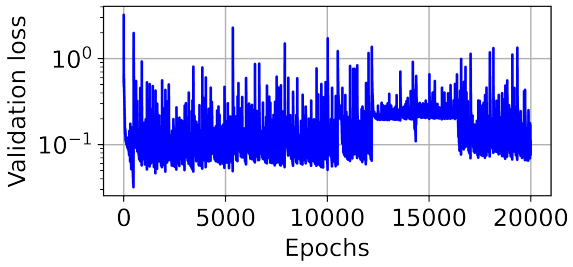


Fig. 5: Validation loss for the full-capacity network.

In Figure 6 and 7 we present the output of the network after training, compared with the FEM solution. The test error is $2.40 \cdot 10^{-4}$, and the final network output is indistinguishable from the FEM output to the naked eye. We see that the main residuals are in the center of the spatial domain, around the shock-wave, where the magnitude of residuals reach 0.23. The value of the PDE as approximated by the network is also at its highest here, where the magnitude is up to 0.47. The training and validation losses are shown in Figure 8. The

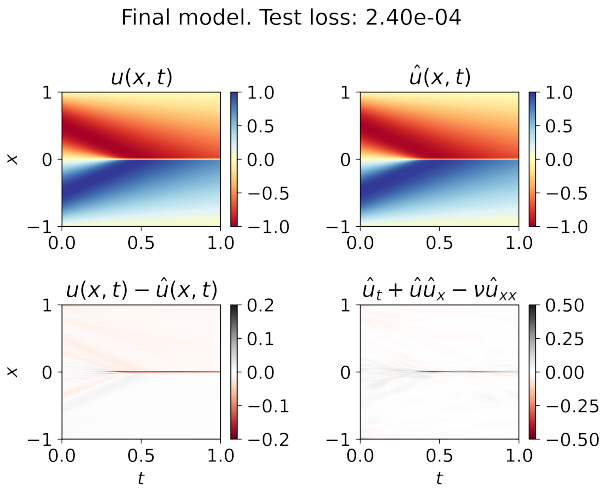


Fig. 6: Output from the final network after 20.000 epochs of training. $u(x,t)$ is the FEM solution, $\hat{u}(x,t)$ is the PINN solution. The bottom panels show the residual between the two, and the value of the PDE as approximated by the network.

two losses are nearly indistinguishable, with minimums of $1.35 \cdot 10^{-4}$.

Parameter	Description
Learning rate	0.005
Hidden layers	4
Neurons per hidden layer	30
Activation function	Hyperbolic tangent
Loss function	Mean squared error
Loss scaling	Not used
Collocation points	Random in each epoch
Number of collocation points	10.000
Number of boundary/initial points	300

Table 1: Description of final network setup.

5. SYSTEM IDENTIFICATION

In system identification it is assumed that some data observations and an underlying model is given, and the goal is then to estimate the true model parameter(s) ν. To investigate how PINNs might also be used for this inverse problem, we sampled 1000 data points from the FEM-solution and initialised the PINN with ν as an unknown parameter of the network. Thus, during training, the network now also updated ν to minimise the loss (2) which included both the data loss and physics losses. We define the ν-loss to be the distance between the network's estimate $\hat{\nu}$ and the true parameter:

$$L_\nu = |\nu - \hat{\nu}|$$

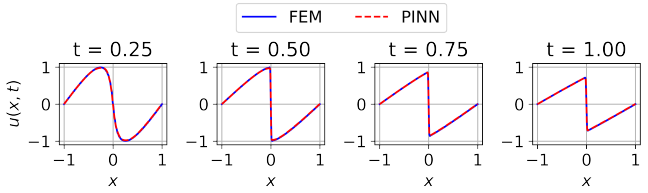


Fig. 7: Cross-section of final network output results compared to the FEM solution.

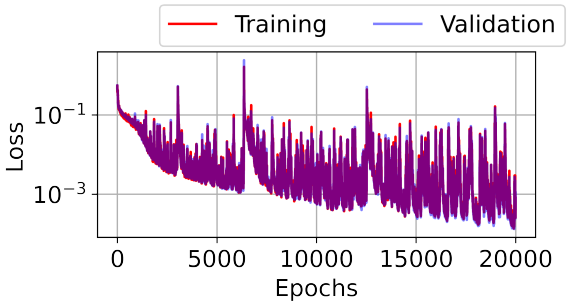


Fig. 8: Training and validation loss for the final network.

The network was initialised with $\hat{\nu}_0 = 1$, while the "true" parameter was in fact $\nu = \frac{1}{100\pi}$. The result is shown in Figure 9. With the high initialisation value of 1 the estimate ended up

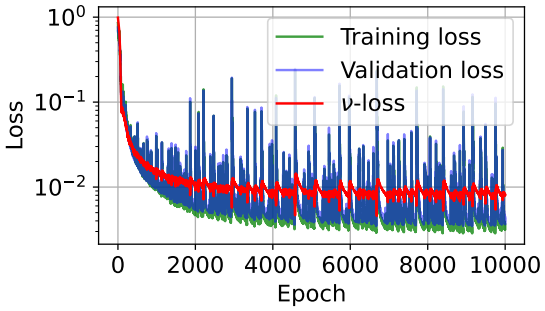


Fig. 9: System Identification.

at a value of $\hat{\nu} = 1.13 \times 10^{-2}$ which is 3.5 times true value $\nu = \frac{1}{100\pi} = 3.18 \times 10^{-3}$. When inspecting the validation loss it is clear that the PINN-solution has converged to a quite accurate estimate $\hat{u}$ of the true solution u. Thus, the $\hat{\nu}$-value found appears to give rise to approximately the same solution and therefore likely represents a local minimum in the parameter space which explains why the PINN has difficulties converging to the true parameter. For some applications, one could argue that a model with the new $\hat{\nu}$-value is acceptable, if it explains the data with the same accuracy as the true latent model.

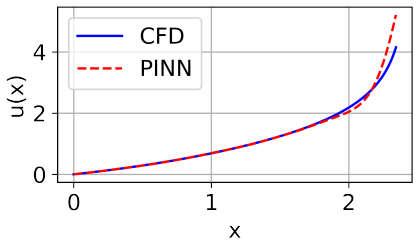


Fig. 10: Output of PINN solution for the SSA, compared with a CFD solution.

6. APPLICATION TO GLACIOLOGY

To investigate the flexilibty of PINNs, we attempt to solve a non-linear differential equation from glaciology. The Shallow Shelf Approximation in its non-dimensional form is ([6])

$$4\epsilon \left(h u_x |u_x|^{\frac{1-n}{n}} \right)_x - (Cu)^{\frac{1}{m}} - h\left(h-b\right)_x = 0, \quad (6)$$

$$u(0) = 0, \quad u_x|_{x=L} = \beta, \quad (7)$$

where the velocity u is a function of the spatial coordinate x, the boundary conditions are Dirichlet and Neumann, and the left of the terms will be left unexplained for brevity. The objective is to solve the function for u for a given topography (h and b), and resistance to flow (C).

To solve the problem, we slightly modify the setup to solving Burger's equation, without tuning the hyperparameters. The output is compared to a CFD solution (see [7]) in Figure 10. The results are qualitatively satisfactory. The BCs are respected, and the overall shape of the network output is similar to the CFD solution. However, a more thorough optimization of the network to solve the task should be performed, before results would be quantitatively useful.

7. DISCUSSION

7.1. Computational efficiency

On a laptop it took around 45 minutes to train the final network, 2.5 hours to train the full capacity network, and 5 minutes to run the FEM simulation. However, the code and framework for the FEM solution is very tailored to the specific problem, whereas we demonstrate that only minor modifications to PINN are necessary in order to get qualitatively acceptable solutions for other problems.

7.2. Improving results

The preliminary analysis of hyperparameters and other network choices could be improved with a grid search, where we could investigate the effect of how different choices affect each other. This is hypothesized to be the reason that the full-capacity network, as motivated by our hyperparameter analysis, performed worse than the final network, which had less neurons per layer. It underlines the challenges in hyperparameter choices in the field. More extensive methods exist, the most simple being exhaustive grid search, random grid search and iterative parameter-by-parameter tuning. This work could further be improved by comparing the analysis of hyperparameters as well as test error to that obtained of others (e.g.[4, 8, 3]).

Furthermore, we see that the PINN is unable to converge to a solution where the PDE is completely satisfied. One way to improve this, could be to implement a collocation sampling method which puts more emphasis on regions of "dramatic" dynamics (e.g. estimated by high values of derivatives or high values of PDE residuals).

Finally we note the possibility that the final PINN is more accurate than the FEM solution used for testing. A different approach to testing a given model, is to use a fabricated solution to Burgers' equation, which would allow comparing the network output to an exact solution.

8. CONCLUSIONS

In this project we implemented a physics-informed neural network (PINN), to solve Burger's equation without training data. The output of the tuned and trained PINN predicted u accurately with a mean squared error of only $2.40 \cdot 10^{-4}$ when compared to a FEM solution. The discrepancies were mainly located around the shock-wave in the center of the spatial domain. We also applied the PINN to the inverse problem of system identification from the FEM-data. Here we obtained a large parameter deviation of 250% but a relatively small validation loss of 4.3×10^{-3} suggesting that we had found an alternative valid parameter for explaining the observed data using Burger's equation. We demonstrated the flexibility of the framework, by applying it to an equation form glaciology with only minor modifications. We find that choices of hyperparameters are interconnected, and thorough methods are necessary for choosing these optimally.

9. REFERENCES

[1] Mayur P. Bonkile, Ashish Awasthi, C. Lakshmi, Vijitha Mukundan, and V. S. Aswin, "A systematic literature review of Burgers' equation with recent advances," *Pramana - Journal of Physics*, vol. 90, no. 6, 6 2018.

[2] A. Ali Heydari, Craig A. Thompson, and Asif Mehmood, "SoftAdapt: Techniques for Adaptive Loss Weighting of Neural Networks with Multi-Part Loss Functions," 12 2019.

[3] Rafael Bischof and Michael Kraus, "Multi-Objective Loss Balancing for Physics-Informed Deep Learning," 10 2021.

[4] M. Raissi, P. Perdikaris, and G. E. Karniadakis, "Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations," *Journal of Computational Physics*, vol. 378, pp. 686–707, 2 2019.

[5] Jared L. Deutsch and Clayton V. Deutsch, "Latin hypercube sampling with multidimensional uniformity," *Journal of Statistical Planning and Inference*, vol. 142, no. 3, pp. 763–772, 3 2012.

[6] Victor C. Tsai, Andrew L. Stewart, and Andrew F. Thompson, "Marine ice-sheet profiles and stability under Coulomb basal conditions," *Journal of Glaciology*, vol. 61, no. 226, pp. 205–215, 5 2015.

[7] Øyvind A. Winton, Sebastian B. Simonsen, Anne M. Solgaard, Robert McNabb, and Nanna B. Karlsson, "Basal stress controls ice-flow variability during a surge cycle of Hagen Bræ, Greenland," *Journal of Glaciology*, pp. 1–15, 11 2021.

[8] Shengze Cai, Zhiping Mao, Zhicheng Wang, Minglang Yin, and George Em Karniadakis, "Physics-informed neural networks (PINNs) for fluid mechanics: A review," 5 2021.