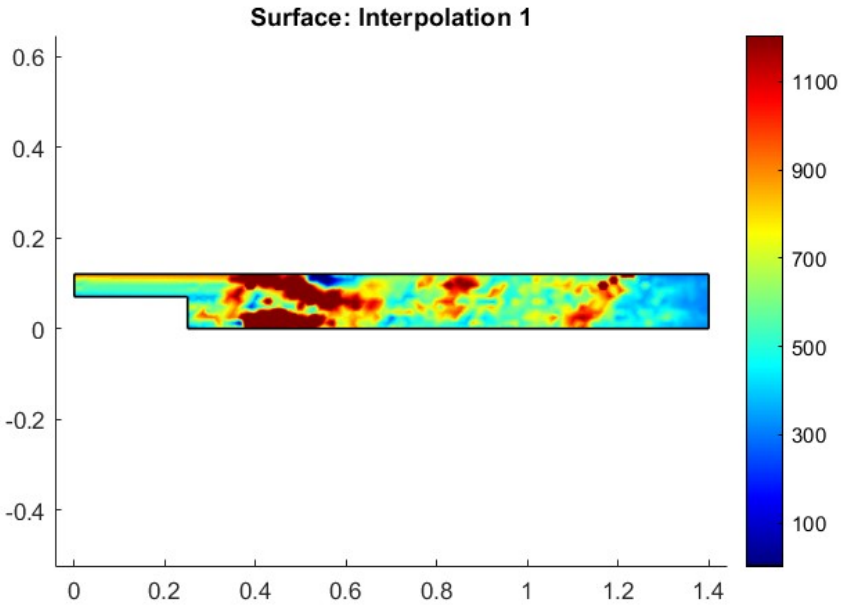
Interim Thesis MSc APPLIED MATHEMATICS

Predicting the Optimal Solver Settings with Machine Learning in the COMSOL CFD Module



Anouk Zandbergen

Technische Universiteit Delft

COMSOL supervisor

Dr. T.L. van Noorden

TU Delft supervisor

Dr. A. Heinlein

June, 2022 Delft

Contents

1 Introduction

The Finite Element Method (FEM) is a popular numerical method for discretizing differential equations related to engineering and modelling problems. Software that use the FEM to solve problems in the fields of, for example, structural analysis and fluid flows are popular among engineers. Within these software it is possible to build models and make simulations, where the difficult discretization and solving phases are performed automatically by the software itself, making it user-friendly. There is a wide range of different FEM software, of which COMSOL Multiphysics is one.

One challenge, however, is the fact that the performance of the solver depends on the solver method and settings. And there are a lot of solver methods like Automatic Newton and Newton Constant, and solver settings like damping factor of the non-linear solver and which initial conditions to use. There are standard solver settings available so that the software user does not have to think about all these settings, however the standard solver settings are not always the best settings for a given problem. In the worst case the solver won't converge with the standard settings, but can converge and give a solution with the right solver settings. Choosing the right solver settings is a challenge, also for experienced engineers, physicists and mathematicians.

The aim of the interim thesis is to investigate the possibilities of a neural network to predict the local CFL number with the use of local data, such that the required iterations to obtain a converged solution is minimized. In this interim thesis, we will look into a back-step laminar fluid flow problem, for which the geometry is shown in figure 1. In this model, the Navier-Stokes equation is discretized and solved by the COMSOL software. We will investigate the possibility of a neural network predicting an optimal local CFL number, in order to improve the convergence and accuracy of the pseudo time stepping method. Local information of the triangular mesh elements is used as an input of this neural network, in order to achieve the generalizability. A data point consist of information of the solution vectors u, v and p and their residuals of the previous time step on each triangle vertex, the Reynolds number on the element and the length of each element edge.

The CFL number is used during pseudo time-stepping, which is a method for solving stationary non linear differential equations. The pseudo-time step itself depends on the CFL number. The higher the CFL number, the closer the method is to Newton's method. This CFL number now only depends on the iteration count or the nonlinear error estimate and thus it is a global expression. Therefore, it might be interesting to investigate if a neural network can predict an optimal and local CFL number that depends on certain solution parameters.

Creating the neural network is done in Matlab. Due to the Matlab livelink for COMSOL, it is easy to obtain information from COMSOL models in Matlab, but also to adapt and run COMSOL models with Matlab commands.

The structure of the interim thesis is as follows: first some related work is discussed in section 2. Secondly, in section 3 the setup of the back-step in COMSOL is explained. Thirdly, background information about Navier-Stokes, pseudo time-stepping and neural networks is given in section 4. Then obtaining optimal CFL number for training and testing neural networks is explained in section 5. Next the data sets and neural networks are discussed in section 6 and their results are discussed in section 7. Lastly, further research on this topic is discussed in section 8

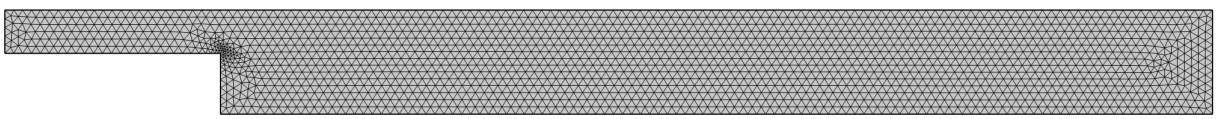


Figure 1: The geometry and mesh of the back-step laminar flow simulation in COMSOL.

2 Related work

Using machine learning and neural networks to replace or enhance existing numerical methods is currently a popular research topic. However, there has not been performed a lot of research on predicting the optimal solver settings with machine learning. Two articles, [6] and [7], will be discussed and will be compared to the research topic of this thesis. In both articles neural networks are used to enhance existing numerical methods and describe how generalizability is achieved. Both articles use local data from the mesh elements in order to achieve this, which we are also going to do for the neural network that predicts the CFL number.

In [6] a network is made that enhances multigrid solvers by letting the network correct interpolated solutions on the fine grids. The network uses local patches, thus it uses local data which results in a generalizable network that can be used for different meshes and flow problems.

First, the Navier-Stokes equations are solved for the first "coarse" layers of the multigrid hierarchy. Then the network predicts a correction of the interpolated solution vector towards the real and unknown solution. This correction is then feed back to the first "coarse" layers and affects the time evolution of the solution, without using the expensive "fine" multigrid layer hierarchy [4].

For training this network, local information is used as an input. That is the residuals, velocities, cell size and Péclet number. The Péclet number is a dimensionless number that gives the ratio of the advection and diffusion rate. With this local information, the loss is then computed as the difference of the solution that uses the correction output of the network and a reference solution. This reference solution is obtained by actually performing the multigrid method with levels of finer grids.

The network reduces the computation time, while obtaining the same or higher accuracy compared to standard multigrid solvers for some of the models in [6]. However, not all of the models showed improvements, thus further research should improve the generalizability of this network.

In [7] a network is built that detects troubled-cells in a mesh. These troubled-cells suffer from Gibbs oscillations as a result of applying high-order numerical methods in regions with discontinuities. Limiting the solution during post-processing can control these oscillations, but these methods also limit in cells that do not show oscillations, making it computationally expensive and reduces the accuracy in smooth regions. Detecting the troubled-cells and only limit them can increase the accuracy and is less computationally expensive.

There already are methods to flag troubled-cells, such as minmon-type TVB (total variation bounded) limiters [7]. However, these methods require problem-dependent parameters. These parameters are determined empirically, and a poor choice of the parameters result in a poor result of the troubled-cell detection. Training a neural network to detect these cell does not require these empirically determined problem-dependent parameters.

The input of the neural network contains local information of the cells, which results in a network that is generalizable. The outcome classifies the cells into the classes "good" or "troubled" cells. Cross entropy is used as the loss function, where the true values are generated by setting it to $[1, 0]^\top$ or "troubled" if a discontinuity or kink appears, and to $[0, 1]^\top$ otherwise.

This method that uses the network outperforms the traditional TVB indicator in term of solution accuracy and number of flagged cells. The computational costs of these methods are comparable. This research is performed with a structured quadrilateral two-dimensional grid, and with these outcomes it is interesting to build a network that can detect troubled-cells on an unstructured or three-dimensional grid.

As we see in [6] and [7], they both use local information for their network in order to create a generalizable network, i.e. a network that can also perform well for different meshes or models of the same kind. As we want to create a neural network that can predict the CFL number for pseudo time-stepping for general Navier-Stokes models in COMSOL, we will also use local information of the model grid. This information will be the solution vector, the residuals, Reynolds number and cell edge lengths.

Furthermore, the articles use a neural network to enhance current numerical methods, by making them computationally more efficient and more accurate in some cases. We want to achieve that the network created in this thesis predicts the best local CFL number such that the pseudo time-stepping converges faster and returns a solution with the same or a higher accuracy than the standard pseudo

time-stepping method in COMSOL.

However, besides these similarities there are also some differences between the networks in [6, 7] and the network in this thesis. First of all, [6] and [7] create a network that replaces one part of the numerical method with a neural network. In [6] they replace the finer grid hierarchy of the multigrid method with a neural network and in [7] they replace the method to detect troubled-cells. The network in this work will determine a parameter used in the pseudo time-step, so in fact it will predict a solution method parameter and will thus not replace (a part of) the method.

Secondly, when computing the loss in [6] and [7], there are reference solutions available. In [6] they use the solution obtained by using the multigrid method including the finer grid hierarchy as a reference and in [7] they knew which cells were troubled by looking if the solution in those cells contained any discontinuity or kink. In this work, we do not have any information about the "best" choice of CFL number. We define the loss with the use of the residuals of the model after one pseudo time-step using the CFL number predictions of the network. The idea is that if the residuals are minimized, then the CFL number outputs of the network perform the best for the pseudo time-step.

Figure 2: The geometry back-step laminar flow simulation in COMSOL, with inflow boundary in red and outflow boundary in blue. The dimensions are given in Table 1.

3 Simulation setup in COMSOL

The simulation created in COMSOL is a backward-facing step. The geometry of this model is shown in figure 2. The inflow boundary is located at the top left edge coloured in red, and the flow moves towards the left outflow boundary coloured in blue. Due to the increase of cross-sectional area, the laminar flow, the Reynolds number increases and flow separation can occur [8]. Laminar flow is more likely to occur with higher flow velocities and a larger difference in cross-sectional area.

In this section the simulation setup in COMSOL will be discussed, such that the model can be reproduced in COMSOL by the reader. The geometry, mesh and elements, study steps and the sensitivity analysis and optimization of the CFL number will be explained.

3.1 Geometry and initial conditions

There are several possibilities for the geometry and the initial conditions when creating a backward-facing step simulation. The chosen geometry does not converge with the default solver settings, but it does when pseudo time-stepping is used.

The dimensions of this backwards-facing step flow simulation are given in table 1. The other factors that influence the solution are the inflow velocity and the initial velocity. The inflow velocity is the speed in which the flow enters the inflow boundary, which is set to 0.03 m/s and is constant in speed and time. It is easier to obtain a converged solution if the initial velocity is around the same as the inflow velocity. However, since we want a boundary case that does converge with a right choice of CFL number and does not converge with a wrong choice of CFL number, the initial velocity is set to 0. With this geometry and initial conditions, the simulation is able to obtain a converged solution as shown in figure 3.

3.2 Mesh and elements

For this model we use triangular elements. In section 6.1, we will see that the information on the element vertices, edges and centroid are used as an input for the neural network. Only working with triangular elements makes it easier to have inputs for the neural network with the same size.

In the COMSOL simulation, we use a coarser mesh size, with maximum element size 0.0104 and minimum element size 4.8E-4. There are no special elements on the boundary layers, because we will then also have quadrilateral elements in the mesh. The default discretization for laminar flow is used, that is P1+P1 (triangular) elements. The P1+P1 elements are piecewise linear elements for both the velocity and pressure [1]. The mesh is shown in figure 1, which inn total has 1630 elements.

Inflow boundary height	0.05 m
Outflow boundary height	0.12 m
Width before increase in height	0.25 m
Width after increase in height	1.15 m

Table 1: Dimensions of the back-step simulation in COMSOL

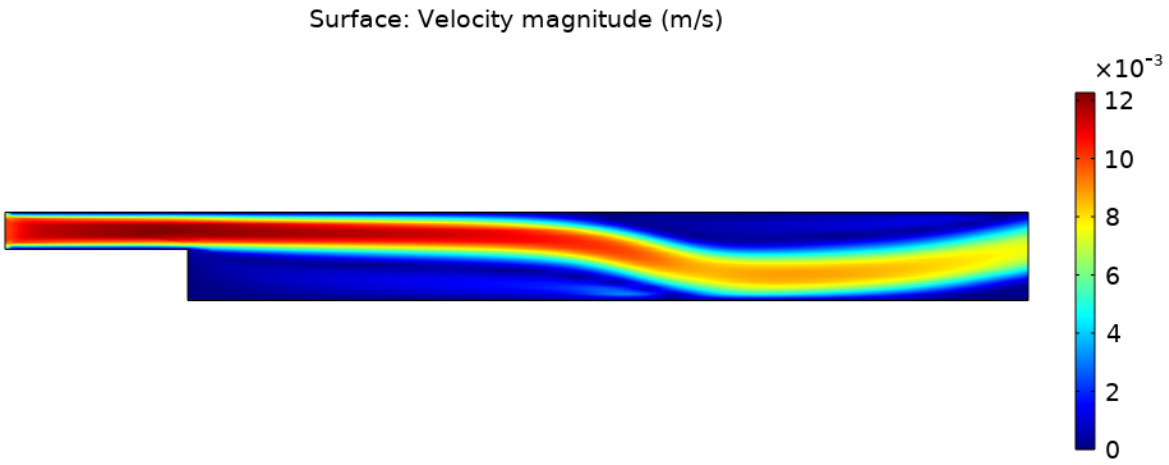


Figure 3: Converged solution of a laminar flow simulation of the back-step model in COMSOL.

3.3 Study steps

An organized plan for the study step is a necessity, since there is a step required to compute the solution with the default pseudo time-step settings, a step that computes the solution with the predicted CFL number, a sensitivity analysis to compute necessary derivatives and an optimization step to compute the optimal value for the CFL number.

First we will discuss the study step with the default settings, then the study step that uses the predicted CFL and how this step is connected to the first study step with default settings, and lastly we discuss the sensitivity analysis and optimization step.

3.3.1 Study step with default settings

First, a study step is created to compute the solution for the first steps using the default settings for pseudo time-stepping. The information of the last step of this study will be used as an input for the neural network, or to create a data set to train the network with. The data set and network inputs will be discussed more thoroughly in section 6.1.

In the laminar flow menu, in advanced settings, the pseudo time-stepping for the stationary equation form is enabled with an automatic CFL number expression. This expression is shown in equation (7). The used solver is Newton constant with a damping factor of 1 and is terminated when a given number of iterations is reached.

The pseudo time-stepping is also used as a stabilization and acceleration technique within the fully coupled stationary solver. The pseudo time-step algorith is explained in secion 4.2 Again, the default settings for this method are used.

3.3.2 Study step with CFL prediction

After the study step that performs some iterations with the default settings, another study step is performed that uses the output of the neural network as the CFL number for the pseudo time-stepping. Besides this, a sensitivity analysis computes the derivative of the loss, defined in section 6.3.2. In this section, we will explain how the laminar flow solution is obtained and in section 3.4 the sensitivity analysis will be explained.

The initial solution for this step is obtained by the last solution of the study step with default settings for the pseudo time-stepping. However, in order to include the CFL number obtained from the neural network in Matlab, an extra laminar flow interface is created to include this CFL number in COMSOL.

First, an interpolation function is defined. This interpolation function reads the output of the neural network combined with its coordinates from a text file and interpolates these values over the domain. Then, a weak contribution is added to the new laminar flow interface. The weak expression is given as

```
spf.rho*(-(u-nojac(u2))*test(u)/cfl2-(v-nojac(v2))*test(v)/cfl2)
```

This is the function that COMSOL uses to compute the pseudo time-step. Here, `cfl2` is the predicted CFL number from the neural network.
The fully coupled solver computes one Newton constant step, with damping factor 1, with the predicted CFL number and pseudo-time stepping.

3.4 Sensitivity analysis

As we will see in section 6.3.3, we also have to perform a sensitivity analysis to obtain a derivative that needs to be computed for the gradient. The sensitivity interface in COMSOL does not contain any physics, but it evaluates the sensitivity of the model with respect to a given variable [1].

Evaluating the sensitivity of a function $F(x)$ with respect to the variable x at a given point x_0 can also be seen as computing the derivative $\partial F/\partial x$ at the point x_0 [1]. We want to obtain the derivatives of $\partial res_u/\partial u$, $\partial res_v/\partial v$ and $\partial res_p/\partial p$ at the points of the solution vector of u, v and p. That means we have to perform sensitivity analysis on the functions res_u, res_v and res_p with respect to the control variables u, v and p.

3.5 Optimization

The neural network has to be trained so that it can predict the CFL number that leads to the smallest loss. In order to understand which local CFL number leads to the smallest loss in COMSOL, an optimization study step is performed within COMSOL.

The general optimization study step minimizes the value of a given global objective for a given control variable field. The global objective is the loss function and the control variable is the cfl number. The initial value for the cfl number is set to 300, the optimization method used is SNOPT with a tolerance of 1E-13 and maximum number of evaluations set to 100,000.

4 Background information

In order for a neural network to make good predictions, it must get useful information to base those predictions on. To decide which information can be useful as a network input, it is necessary to know which mathematical factors can play a role in the CFL number choice.

The neural network uses the input values to make a prediction. The network will be trained to learn which prediction is best. For the training, backpropagation is used.

First, we have a short introduction to the Navier-Stokes equations. The backwards-facing step model explained in section 3, describes a laminar fluid flow problem. These are described by the Navier-Stokes equations and are solved by COMSOL with the use of pseudo time-stepping.

Secondly, we discuss the pseudo time-stepping algorithm and the definition of the local CFL number that plays a role in this algorithm. With the neural network, we want to predict the optimal CFL number for the pseudo time-stepping method. We then see which factors play a role in this algorithm and based on this information we can determine the input parameters for the neural network.

Lastly, necessary information about neural networks is given. Here the multi-layer perceptron and the backprogagation algorithm will be explained.

4.1 Navier-Stokes equations

The Navier-Stokes equations are nonlinear partial differential equations that describe viscous flow. In the problem with the back-step simulation, we consider incompressible flow. That is a flow in which the density of each material particle remains the same [10]. The stationary Navier-Stokes equations for incompressible flow are given by

$$\begin{aligned} \rho_0(\mathbf{u}\cdot\nabla)\mathbf{u} - \mu\nabla^2\mathbf{u} &= -\nabla p + \mathbf{f}^b, \\ \rho_0\nabla\cdot\mathbf{u} &= 0. \end{aligned} \tag{1}$$

In these equations, $\mathbf{u}$ is the flow velocity, μ is the viscosity of the fluid, ρ_0 is the density which is assumed to be constant, p is the pressure and $\mathbf{f}^b$ is the body force acting on the flow [10].

In COMSOL, we use the laminar flow physics interface. As explained in the COMSOL reference manual, a flow remains laminar as long as the Reynolds number is below a critical value [1]. The Reynolds number Re is given by

$$\mathrm{Re} = \frac{\rho U L}{\mu}. \tag{2}$$

Here, U and L are the typical velocity and length scales. The Reynolds number gives the ratio between inertial and viscous forces [1]. At low Reynolds numbers, the flow remains laminar as the viscous forces are able to damp out disturbances in the flow. At high Reynolds numbers, the inertial forces become more important resulting in nonlinear interactions to grow, which causes turbulence. The Reynolds numbers in the back-step flow model have values between 0 and 10.

4.2 Pseudo time-stepping

Solving stationary nonlinear differential equations with Newtons method only, requires an initial guess close enough to the root. If this initial guess is not available, Newtons method is not able to give a solution. Computing the steady state by solving the fully transient problem will be able to give a result, but this method is known as costly in terms of computational time [1, 3].

Pseudo time-stepping is a less time costly method that will lead to a solution more often [3]. The method will frame the problem in a time-depending setting:

$$\frac{\partial \mathbf{u}}{\partial t} = F(\mathbf{u}). \tag{3}$$

with $F(\mathbf{u})$ the differential equation of which the steady state solution must be determined. In [3] the algorithm for pseudo time-stepping is described as follows

$$\frac{\partial \mathbf{u}}{\partial t} = -V^{-1}F(\mathbf{u}), \quad \mathbf{u}(0) = \mathbf{u}_0. \tag{4}$$

Where V is a nonsingular matrix that scales the problem. The sequence of iterations is given by

$$\mathbf{u}_{n+1} = \mathbf{u}_n - \left(\frac{V}{\Delta\tilde{t}_n} + F'(\mathbf{u}_n)\right)^{-1} F(\mathbf{u}_n). \tag{5}$$

With $F'(\mathbf{u}_n)$ the Jacobian. The pseudo code is given in Algorithm 1 [3]. In COMSOL [1] the pseudo time-step is defined as

$$\Delta\tilde{t}_n = \mathrm{CFL}_{\mathrm{loc}}(n)\frac{h}{\|\mathbf{u}\|}, \tag{6}$$

with h the mesh cell size and $\mathrm{CFL}_{\mathrm{loc}}$ the local CFL number [1]. The method starts with a small CFL number and gradually increases it as the solution reaches convergence [1]. When the pseudo time-step approaches infinity, the method will behave as a normal Newton step.

There are two options for the CFL number within COMSOL. In the first option, the local CFL number used in COMSOL depends on the iteration count of the nonlinear solver n:

$$\mathrm{CFL}_{\mathrm{loc}}(n) = 1.3^{\min(n,9)} + \mathrm{if}(n > 20, 9 \cdot 1.3^{\min(n-20,9)}, 0) + \mathrm{if}(n > 40, 90 \cdot 1.3^{\min(n-40,9)}, 0). \tag{7}$$

In the second option, the CFL number depends on the nonlinear error estimate e_n for step n, the the given target error estimate "tol" and the controller parameters k_P, k_I and k_D, which are positive constants:

$$\mathrm{CFL}_{n+1} = \left(\frac{e_{n-1}}{e_n}\right)^{k_P} \left(\frac{\mathrm{tol}}{e_n}\right)^{k_I} \left(\frac{e_{n-1}/e_n}{e_{n-2}/e_{n-1}}\right)^{k_D} \mathrm{CFL}_n. \tag{8}$$

For this CFL number a hard lower limit $\mathrm{CFL}_n \geq 1$ is used and convergence is accepted when $\mathrm{CFL}_n \geq \mathrm{CFL}_\infty = 10^4$.

As we can see in equations (7) and (8), the local CFL number is a global expression as it is the same for each mesh element. The pseudo time-step however does depend on the size of the mesh cell, making the time-step itself a local expression.

Since the local CFL number only depends on the number of iterations and is the same for each mesh cell, it is interesting to investigate if there is an optimal value for $\mathrm{CFL}_{\mathrm{loc}}$. One could think of adding information about the solution, residuals, mesh and the Reynolds number. A neural network can have all this information as a input and can then output a CFL number for each mesh cell, resulting in a local CFL number that takes all this information into account and is actually a local value.

Algorithm 1 Pseudo time-stepping

1: Set $\mathbf{u} = \mathbf{u}_0$ and $\Delta\tilde{t} = \Delta\tilde{t}_0$.
2: **while** $\|F(\mathbf{u})\|$ is too large **do**
3: Solve $(\Delta\tilde{t}_n^{-1} I + F'(\mathbf{u}))s = -F(\mathbf{u})$
4: Set $\mathbf{u} = \mathbf{u} + \mathbf{s}$
5: Evaluate $F(\mathbf{u})$
6: Update $\Delta\tilde{t}_n$
7: **end while**

4.3 Neural networks

An artificial neural network is a deep learning method that is inspired by how our own human nervous system processes information and has been introduced around the 1940s [2, 9]. The interest of using and investigating neural networks has been up and down. Improvements of the network architectures led to an increasing interest in the early 1980s, but when other alternative machine learning methods, like Support Vector Machines (SVM)'s, were created which seemed to perform better, there was a decreased interest in the 1990s [2]. Right now, neural networks are again a hot research topic in several fields of engineering, mathematics and computer science. In [2] some reasons are summed up:

- Availability of a huge quantity of data, where neural network mostly outperform other machine learning techniques;

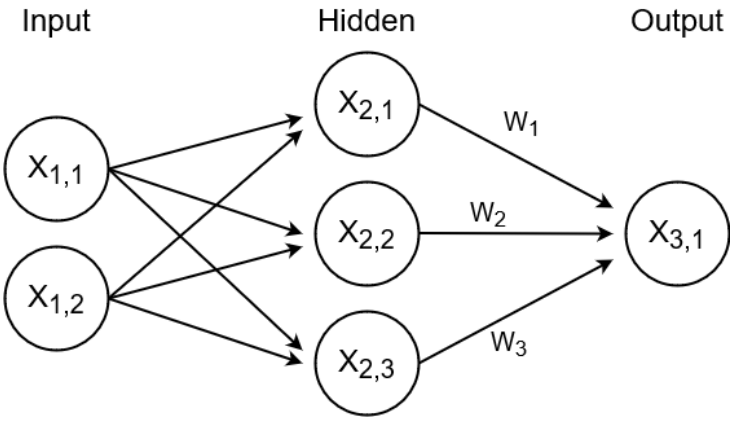


Figure 4: An MLP with two input neurons, three neurons in the hidden layer and one output neuron. The weights between the hidden layer and the output layer are shown, but are left away between the input and hidden layer for simplicity.

- Increased computing power due to improved GPU cards and Moore's Law, resulting in a reasonable training time for the neural network;
- Improved training algorithms for neural networks;
- There is more attention to successful neural network studies, resulting in more interest and curiosity to investigate the possibilities of neural networks;
- Availability of software and manuals that make programming, creating and using neural networks relatively easy.

In this thesis, a neural network will be created to investigate if it can predict the optimal CFL number, resulting in an optimal performing pseudo time-step. That is, with this optimal CFL number, the pseudo time-step algorithm in COMSOL will converge faster with this optimal CFL number.

4.3.1 Multi-Layer Perceptron

The network we will build has multiple hidden layers and is therefore a Multi-Layer Perceptron (MLP). An MLP has an input layer that passes through the information to the first hidden layer. If the network has two or more hidden layers, it is often referred to as a deep neural network or DNN. Finally, the last hidden layer passes its information to the output layer [2]. In this thesis, the output layer will output the local CFL number.

Each layer consists of neurons, and each neuron in a layer is connected to all the neurons in the next layer, as shown in figure 4. In this figure we can see that each neuron in the input layer has a connection to all of the neurons in the hidden layer. Each of these connections has a certain weight w_i, which is shown on the connections between the hidden layer neurons and the output layer neuron. During training, these weights are adapted in order for the network to learn a certain task.

We can mathematically describe how information is passed from one layer to another. We will use figure 4 as the example network. From the input layer to the hidden layer we see six connections and each connection has its own weight. Furthermore a bias b is added to possibly shift the activation function. This activation function maps the output to a certain range and ensures that the learning algorithm will be able to make progress in each step [2].

The information from the input neurons is passed through the neurons in the hidden layer in the following way [9]:

$$\begin{aligned}
\sigma_1(w_{1,1}x_{1,1} + w_{1,2}x_{1,2} + b_1) &= x_{2,1}, \\
\sigma_1(w_{2,1}x_{1,1} + w_{2,2}x_{1,2} + b_2) &= x_{2,2}, \\
\sigma_1(w_{3,1}x_{1,1} + w_{3,2}x_{1,2} + b_3) &= x_{2,3}.
\end{aligned}$$

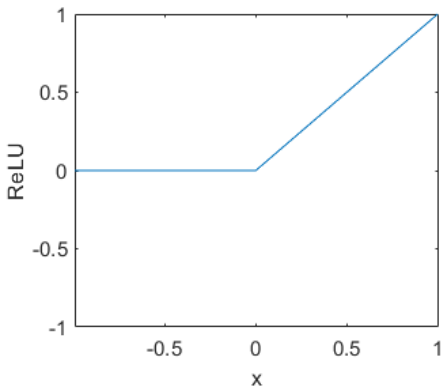


Figure 5: $\mathrm{ReLU}(x) = \max(0, x)$ activation function.

Here, σ_1 is the activation function, $w_{i,j}$ are the weights of the connections between the input layer and the hidden layer and $x_{i,j}$ are the values of each neuron shown in figure 4. The network learns a task by adapting the weights each iteration by following the backpropagation algorithm.

The activation function that we use is ReLU, which is defined as

$$\mathrm{ReLU}(x) = \max(0, x).$$

This function is not differentiable at $x = 0$, however this activation function works very well, is cheap and efficient, and does not show any problems when training the neural network[2].

In the last layer of the network, there is no activation function. The CFL function can be any real value except zero. Therefore, by omitting the activation function in the last layer, the network can output any real number.

4.3.2 Backpropagation

Backpropagation is used by gradient descent and is an optimization algorithm used for training neural networks. The network weights are adapted each training step so that a so-called cost function or loss is minimized [2]. The loss is a function that measures how good the network predictions are. The mean squared error (MSE) is for example a loss function, so naturally when the MSE is minimal the networks predictions are the most accurate.

Gradient descent measures the local gradient of the loss with respect to the predicted value of the network, in our case that would be the local CFL number, and moves towards the direction of the descending gradient [2]. That means we have to compute the derivative of the loss function with respect to the learnable parameter in order to apply backpropagation. This derivative is propagated backwards into the neural network in order to train the network, that is where the name comes from. This means that this procedure first determines how much each individual neuron in the last hidden layer contributed to the error, then it propagates backwards into the network to determine the contribution of each layer to the error, until the input layer is reached. Finally, the algorithm adapts the network weights to reduce the loss [2].

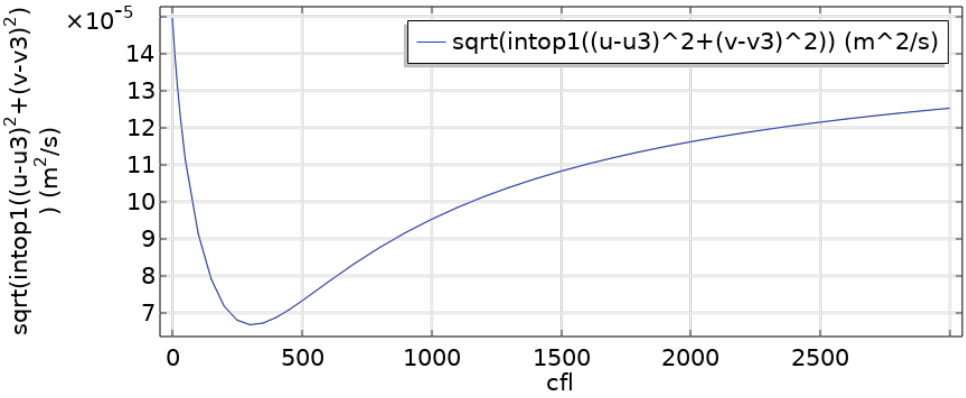


Figure 6: The outcome of the parametric sweep study for global CFL numbers ranging from 0 to 3000. The minimum of the loss is around a CFL number of 300.

5 Optimal CFL numbers

The neural network will be trained to predict the optimal CFL number. The optimal CFL will be the CFL number that results in the minimal value for the loss function (11). The prediction is that the CFL number that minimizes this loss will result in a solution that converged faster than the default settings for pseudo time-stepping.

The CFL number that minimizes the loss function in (11) can be found with existing optimizers in COMSOL or in Matlab. In COMSOL we will both look at the optimum for the global CFL and for the local CFL, and in Matlab we will compute the local optimum with the Adam optimizer.

5.1 The optimal global CFL in COMSOL

Finding the optimal global CFL number is done with a parametric sweep in COMSOL. The parametric sweep computes one Newton step with different CFL numbers. The CFL numbers go from 0 to 40 in steps of 10, 50 to 500 in steps of 50 and from 500 to 3000 in steps of 100. For each CFL number the loss given in equation (11) is computed.

The result of this parametric sweep is shown in figure 6. We can clearly see there is one global optimum given around a CFL number of 300, with a loss of 6.68E-5. That means this global CFL number should be relatively easy to predict by a neural network. In the next sections we will compare this result with the optimization results for local CFL values. This optimal global CFL value will furthermore be used as an initial condition for the Adam optimizer in section 5.2.1.

5.2 The optimal local CFL

To get to know whether or not the neural network outputs the optimal CFL, the optimal CFL number can be found with existing optimization algorithms. Within COMSOL the optimization tool is used to find the optimal CFL number for a given step of the backstep simulation, and within Matlab Adam is implemented.

For both optimizers, the loss will be the objective function that has to be minimized and the CFL number is the control variable. That means that the optimizers will find the CFL number that minimizes the loss function in equation (11). With the outcomes of these optimizers we can see what the optimal local CFL number is and we can use it to determine which inputs of the network contribute to a correct prediction.

5.2.1 Adam optimizer in Matlab

Adam is short for *adaptive moment estimation* and is faster than the regular gradient descent algorithm [2]. In the standard gradient descent algorithm, the current value of the CFL number is updated with

the use of learning rate α and the gradient:

$$\mathrm{CFL}_{n+1} = \mathrm{CFL}_n - \alpha \frac{\partial loss}{\partial \mathrm{CFL}}. \tag{9}$$

There are several improved versions of this method which are more likely to converge to the global minimum and with less iterations required, of which one of them is Adam. Adam is a combination of two of these optimizers, namely the momentum method and RMSprop [2].

In gradient descent with momentum, the current gradient is replaced by a momentum. The previous gradients play a role within this algorithm, where the momentum of this algorithm is computed as

$$\begin{aligned} m_{n+1} &= \beta_1 m_n - (1-\beta_1)\frac{\partial loss}{\partial \mathrm{CFL}}; \\ \mathrm{CFL}_{n+1} &= \mathrm{CFL}_n - \alpha m_{n+1}. \end{aligned}$$

The initial value for the momentum is zero. The parameter β_1 is the momentum friction [2]. The momentum uses the average of the past gradients to speed up the vanilla gradient descent method.

RMSprop stand for *Root Mean Squared prop*, and the learning rate α is divided by a parameter v. This method is given as:

$$\begin{aligned} v_{n+1} &= \beta_2 v_n + (1-\beta_2)\cdot\left(\frac{\partial loss}{\partial \mathrm{CFL}}\right)^2; \\ \mathrm{CFL}_{n+1} &= \mathrm{CFL} - \frac{\alpha}{\sqrt{v_{n+1}+\epsilon}}\cdot\frac{\partial loss}{\partial \mathrm{CFL}}. \end{aligned}$$

Where the parameters β_2 and ϵ are respectively the decay rate and a very small number to prevent division by zero. The parameter v is used to speed up the algorithm when the gradients are small, by using the past squared gradients [2].

As mentioned earlier, Adam is a combination of gradient descent with momentum and RMSprop. That means it uses both the momentum m and the vector v. The CFL number update is computed as follows:

$$\mathrm{CFL}_{n+1} = \mathrm{CFL}_n - \frac{\alpha}{\sqrt{\hat{v}_{n+1}}+\epsilon}\cdot\hat{m}_{n+1}. \tag{10}$$

We see $\hat{v}$ and $\hat{m}$, which are the bias corrections of v and m computed as

$$\begin{aligned} \hat{m}_{n+1} &= \frac{m_{n+1}}{1-\beta_1^{n+1}}; \\ \hat{v}_{n+1} &= \frac{v_{n+1}}{1-\beta_2^{n+1}}. \end{aligned}$$

And again, the moment m and the vector v are computed the same as in gradient descent with momentum and RMSprop [2]:

$$\begin{aligned} m_{n+1} &= \beta_1 m_n + (1-\beta_1)\cdot\frac{\partial loss}{\partial \mathrm{CFL}}; \\ v_{n+1} &= \beta_2 v_n + (1-\beta_2)\cdot\left(\frac{\partial loss}{\partial \mathrm{CFL}}\right)^2. \end{aligned}$$

Adam was implemented in Matlab and applied on the backstep simulation in COMSOL to predict the optimal CFL number for the third step of the non-linear solver.

The Adam optimizer started with a learning rate α of 0.7, $\beta_1 = 0.9$ and $\beta_2 = 0.999$ and an initial CFL of 300. Due to the small gradients, a high initial learning rate was chosen and the initial is chosen to be around the optimal global CFL.

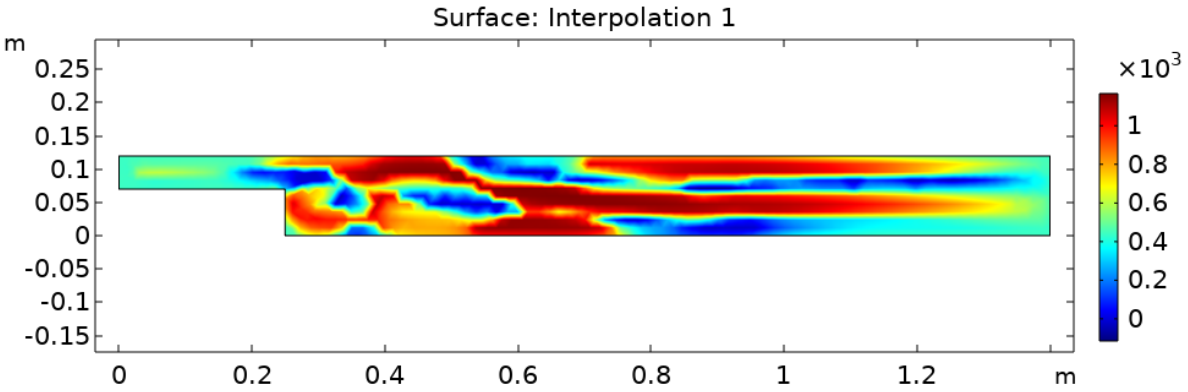


Figure 7: The CFL output of the Adam optimizer, where the loss was minimized to a value of 5.635E-4.

When the algorithm showed (heavy) oscillations during training or the loss would not decrease anymore, the CFL number which resulted in the smallest loss was picked as an initial condition and the learning rate was decreased. By doing so, the algorithm gradually converged to the minimum of the loss, which was 5.635E-4. The outcome of CFL values of the optimizer is shown in figure 7. The final learning rate used was 0.001, where the output of Adam would always result in the same CFL numbers and the same value for the loss.

5.2.2 Optimization study in COMSOL

Within COMSOL there is the possibility to use the optimization study, as discussed in section 3.5. This tool aims to find the minimum of a given global objective for a control variable field. The global objective in this case is the loss in equation (11) and the control variable field is the CFL number. That means COMSOL tries to find the CFL number that leads to the minimum of the loss.

In about twenty minutes the optimization solver finished the computation and obtained a solution. The CFL number obtained by the optimization study is given in figure 8. With these local CFL numbers the loss was minimized to a value of 3.117E-5. This value could even be more minimized with different settings for the optimization study, but it will take significantly longer. Right now, this result is sufficient to get an idea how the optimal local CFL number would approximately look like, to compare with the result of the Adam optimization and to use as a target for the neural network.

When we compare the result of Adam in figure 7 and the result of the COMSOL optimizer in figure 8, we can clearly see that the structures are the same. They both show similarities in the negative areas as well as the areas where the local CFL is relatively high. That probably means the Adam optimizer probably hasn't converged yet. At the same time, the COMSOL optimizer takes less time to obtain an accurate solution.

Comparing the results of the local CFL numbers with the optimal global CFL number, we see that the local CFL number obtained by the COMSOL optimizer result in the smallest loss. Furthermore, when comparing this result with the result of the global CFL, the CFL values of in the area around the inlet and outlet of the local CFL is around 300, which is the optimal global CFL. The solution of the local COMSOL optimizer will be used to train a neural network with this optimal local CFL number as a target.

The aim of using the outcome of the optimization is to investigate which of the current input values contribute to a correct prediction of the neural network. To train this network, the data sets mentioned in section 6.1, and the target or optimized CFL number is created. The loss will then be the Mean Square Error (MSE) of the network output and the target CFL number. Since the computation of this loss will not require any interface with COMSOL, we don't need a custom regression layer, resulting in a faster training of the network. More details about this network will be discussed in section 6.2.

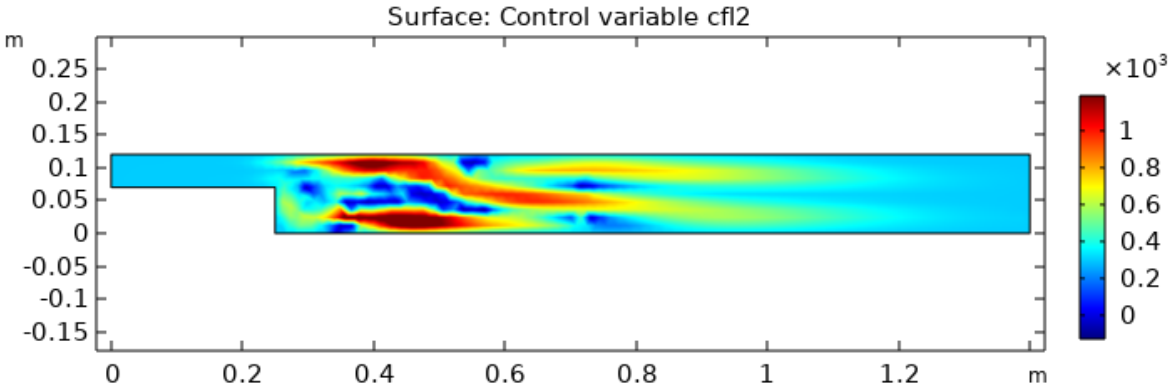


Figure 8: The CFL output of the optimization study in COMSOL, where the loss was minimized to a value of 3.117E-5.

6 Neural Networks to predict the optimal CFL number

Until now two networks with each a different target and loss are created for this research. The first network uses the optimal CFL numbers computed in section 5 as a target for the RMSE loss, and where the second network directly uses he CFL number prediction as an input for the COMSOL simulation to compute the residuals used to compute the loss.

The first network will be used to investigate if the network is able to learn the optimal CFL numbers with the given inputs. If the network is able to learn these optimal CFL numbers, the inputs can be used for the second network. Since this training does not require any interface with COMSOL, as the data set is generated beforehand, the training of the network is relatively fast.

The second network will be used to investigate if it is possible to train the network with a loss computed in COMSOL, where the predicted CFL numbers of the network are plugged in. If this is possible, there is a higher change of generalizability of the network, since no pre-computed target is required to train the network.

Two different data sets will be considered for the training of both networks. One data set will contain information of one element only, where the information of that element is located on the vertices. The second data set will contain information of a patch that contains four elements, and the information will again be located on the vertices of each one of the four elements. Possible benefits of using a patch compared to a single element will be investigated.

6.1 Data sets

For the training of the networks, two different data sets will be considered. The data sets contain local information of the previous Newton step of the COMSOL simulation, where in one set only the data on one element will be considered and in the other set information of a patch of four elements will be available for training. The information of each data point is located on the vertices of each element.

The choice of using local information for the network inputs has several benefits. Firstly, it improves generalizability of the network, making it better applicable for different kind of Navier-Stokes flow problems [6]. The unique model specific information is not used, but only information that is locally available for all laminar flow models is used. This can result in a trained neural network that is applicable for different kinds of laminar flow models.

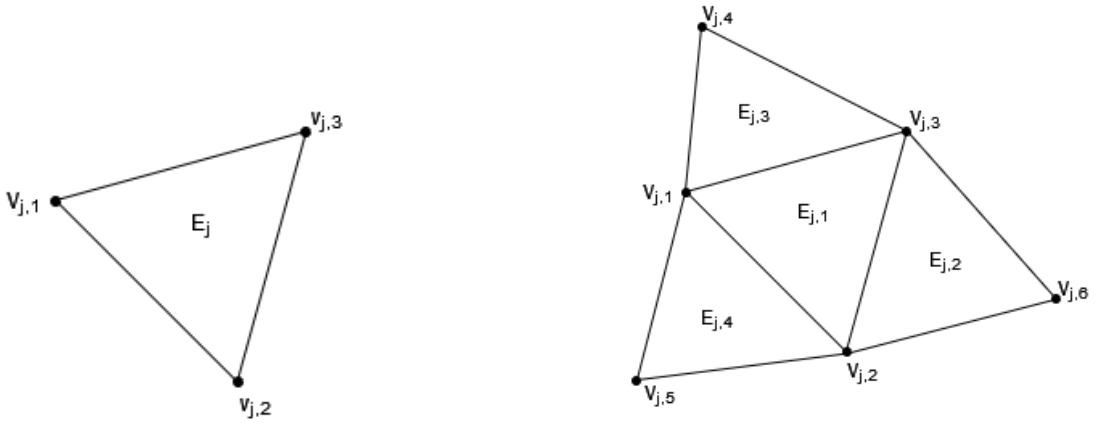
Secondly, only one flow model can generate enough data points for training the network, as one single flow model already contains hundreds of mesh elements [6].

6.1.1 A single element as data point

Introduce a triangular mesh element schematically shown in Figure 9a. This element has centroid E_i and three vertices $V_{j,1}$, $V_{j,2}$ and $V_{j,3}$. Each vertex contains information about the solution vectors of u,

v and p of the current step, as well as information about their two different residuals. The edge lengths of the element and the mean Reynolds number are also used as an input of the neural network, resulting in a total of 31 input variables. The solution vectors and residuals directly give information about the solution and convergence of the simulation and the Reynolds number gives information about the flow. The element edge length is added since the pseudo time-step in equation (6) also contains information about the element size. Furthermore, the element edge length and not the element size is used because the length also contains information of the distance between the vertices.

To combine information of more than one Newton step of the same COMSOL simulation, an extra input variable is added: the iteration number. In the local CFL number in COMSOL the iteration count of the Newton step is included in the calculation of the CFL number. Therefore, this iteration number might give extra necessary information to train the network. The data set that contains information of more than one Newton step therefore has 32 inputs. The data set that contains information of four Newton steps has in total 6517 data points.



(a) Data point which contains only information of one element E_j with vertices $V_{j,1}$, $V_{j,2}$ and $V_{j,3}$

(b) Data point which contains information of a patch of four elements $E_{j,1}$ to $E_{j,4}$ with their vertices $V_{j,1}$ to $V_{j,6}$

Figure 9: Schematic figure of the two different data points.

6.1.2 A patch as a data point

Using local information may be beneficial for the generalizability, but information that is too local can have the disadvantage that it might not contain enough information for the network to be able to learn the optimal CFL. That is why a data set with data points that contains information of a patch of elements will also be created and its effectiveness will be investigated. The patch contains four elements, where one element is the centre element and the other three are its neighbouring elements, as we can see in Figure 9b.

Information of all the vertices of the elements in the patch will be in a data point. The data point order will first contain all the information of the central mesh element $E_{j,1}$, then all the information of element $E_{j,2}$ and so on to element $E_{j,4}$. The information of each element will be the same as the element information in section 6.1.1. This means the information on the vertices $V_{j,1}$ to $V_{j,3}$ are in the data point three times. This seems like it is redundant information, however it might also add extra information on how the elements are connected to each other.

Since the newton step is the same for all the elements within the patch, we will have $4 \cdot 31 = 124$ inputs from the patch and in total 125 per data point if the Newton step is added. The data set that contains information of four Newton steps has in total 6336 data points.

6.2 Network with optimized CFL target

For this network, the data set not only contains inputs for the network and information of the mesh centroid coordinated, but also a desired network output called the target. The aim of this network is to learn to make predictions close to this target. This target is the optimized CFL number in section 5.2.2, with the objective to minimize the loss in equation (11).

First we will discuss the network structure and loss, followed by the results obtained by the neural network.

6.2.1 Network structure and loss

This is a simple network with one input layer, several hidden layer and one output. The input layer contains 32 neurons if the data set that only contains information of one element per data point is used, and 125 neurons if the data set with patches is used. The hidden layers follow the input layer. The optimal number of hidden layers still needs to be determined, but right now we start working with 6 hidden layers. The number of neurons per hidden layer has to be larger than the number of inputs. When there are only 32 inputs, the number of neurons per hidden layers are 128, and if the number of inputs is 125, the number of neurons will be 256. The output layer only contains one neuron to predict the CFL number.

Besides these network parameters, we should also consider the learning rate. When the learning rate is too small, the training of the network take a long time. But on the other hand, when the learning rate is too large, the network outputs during the training oscillate and become too large. Right now, a learning rate of 0.005 is used.

The loss of the network is very simple. We have the target CFL numbers computed in section 5.2.2 and we will use these targets to compute the Root Mean Squared Error (RMSE) between the prediction and the target. The objective of the training of the network will then be to minimize this RMSE.

6.3 Network with objective to minimize residuals

With this neural network, the loss is computed by directly putting in the predicted CFL number in COMSOL and compute the resulting residuals of the study step. That means in every iteration of the network training process, a step in COMSOL is performed. In order to be able to do this, a custom regression layer is built that computes the custom loss and its network gradients with the results from the COMSOL step.

First we will discuss the network inputs that are used for the prediction of the local CFL number. Secondly the output of the network will be discussed, which is the predicted CFL number and some extra information necessary for the computation of the network loss. Lastly the computation of the network loss and gradient in the custom regression layer are explained.

6.3.1 Network structure

The structure of this network is schematically shown in Figure 10, where it is shown that this network has two input layers. The first input layer is has the same purpose as the network in section 6.2, namely to give the neural network information to learn and predict the optimal CFL number. This input layer has ether 23 inputs or 125, depending which of the data sets in section 6.1 is used.

The second input layer with only two inputs has another purpose. Since the network CFL predictions has to be put back into COMSOL again, COMSOL must know the correct coordinates of the corresponding CFL predictions. In order to add information about the centroid coordinated, this input layer is introduced which is directly connected to the output layer.

The output of the network gives a prediction of the CFL number on the mesh centroid x_i. The next pseudo time-step is then performed with the new CFL number and the solutions and residuals on the element vertices are obtained. Since COMSOL can only output values on the mesh vertices within Matlab, the solutions and residuals are interpolated in Matlab in order to obtain the values on the mesh centroid.

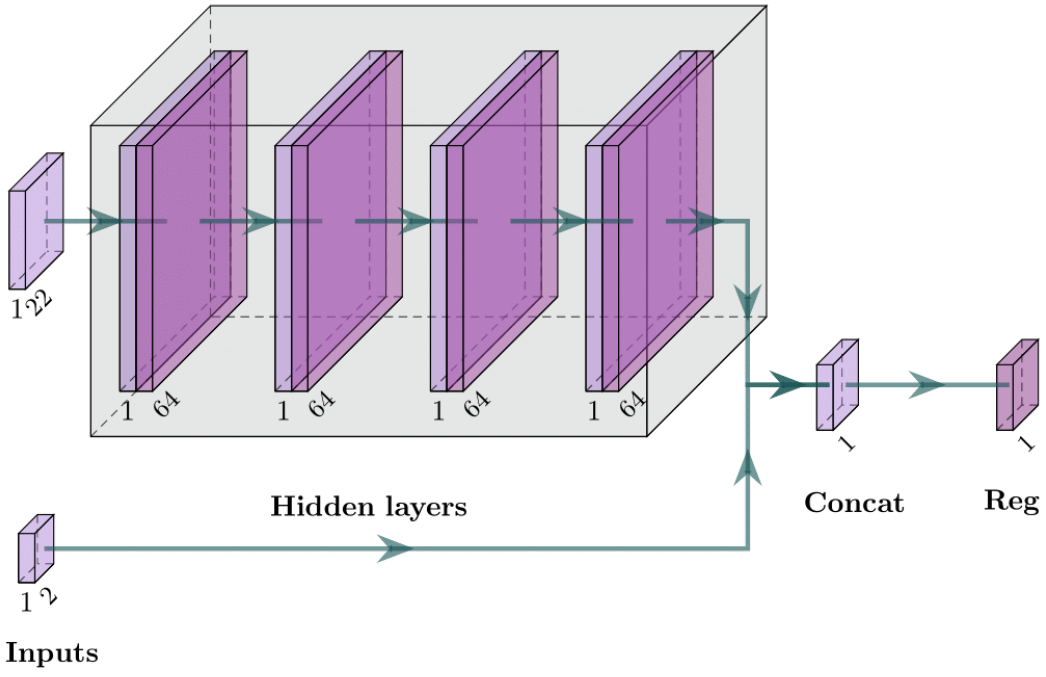


Figure 10: Neural network with the two input layers, hidden layers with 64 neurons, a concatenation layer that connects output of the hidden layers with the second input layer, and finally the custom regression layer. Each fully connected hidden layer (lilac) is connected with a ReLU layer (purple).

With this information of the values on the mesh centroid, the loss and the gradient can be determined. These are computed in a custom regression output layer.

The custom regression layer determines the loss as defined in section 6.3.2 and the gradients as defined in section 6.3.3. Since the output of the neural network, which is the predicted local CFL number, is used as an input in COMSOL to compute the loss, the loss itself is not "traceable" anymore by Matlab. That means that Matlab cannot compute the gradients based on the custom loss function. Therefore, the gradients and the loss should be specified in this custom regression output layer. In order to be able to put the predicted CFL numbers on the correct coordinates, the information of the coordinates given by the second input layer is used.

6.3.2 Network loss

In order to indicate how well the model predicts the CFL number, a loss function should be defined. We want the model to predict the local CFL numbers such that the CFD simulation in COMSOL converges faster, so minimizing a function that estimates the error could possibly lead to better CFL predictions. In order to create the loss, first the converged solution is computed in COMSOL. These "exact" solutions are called u_{ex} and v_{ex}. Applying the means squared error on the solutions u and v computed with the predicted CFL number and the converged solution will lead to a loss defined as

$$loss = \left(\int_{\Omega} (u - u_{ex})^2 + (v - v_{ex})^2 \, \mathrm{d}\Omega \right)^{\frac{1}{2}}. \tag{11}$$

This loss function does not take into account that CFL numbers of 0 are not allowed. In order to create a loss that penalizes small CFL numbers, an extra term is added. This term vanishes as the CFL number increases.

$$loss = \left(\int_{\Omega} (u - u_{ex})^2 + (v - v_{ex})^2 \, \mathrm{d}\Omega \right)^{\frac{1}{2}} + \text{1E-6} \cdot \sum \frac{1}{\text{CFL} + \text{1E-8}} \tag{12}$$

With this loss, CFL numbers near zero are penalized and the bigger the CFL number gets, the less influence this extra term has on the total loss. This factor especially play a role at the beginning of training the network, when the outputs of the network are below 1.

For training the network with a simulation in COMSOL of which we know the converged solution, this loss function can be used. However, if we want to further train the network with other COMSOL simulations of which we do not know the converged solution, another loss should be chosen. So for generalization, this loss function is not ideal. For now, we use this loss function to see if it is possible for this model to predict the optimal CFL number.

6.3.3 Network gradients

We want to train the network using the loss computed in the previous section. Normally, the programming language computes the gradients of this network loss itself. However, since this is a very specific loss function that requires a computation step in COMSOL, the Matlab is not able to compute the gradients of the loss itself.

In order to compute the network gradients, the derivative of the loss function with respect to the CFL number needs to be computed. Luckily, COMSOL can compute the derivative of the first term and the derivative of the second term is not difficult to determine.

The derivative of the first term can be computed within COMSOL using a sensitivity analysis. The global objective in this analysis will be the first term in the loss, given in equation (11). The control variable will be the predicted CFL numbers, using the interpolation function with the predictions as an initial value. All that has to be done to compute this derivative is updating the interpolation function data source, run the simulation for one step and obtain the results from the sensitivity analysis.

The derivative of the second part of the loss is easy to obtain:

$$\frac{\partial}{\partial \mathrm{CFL}} 1\text{E-}6 \cdot \sum \frac{1}{\mathrm{CFL} + 1\text{E-}8} = -1\text{E-}6 \cdot \sum \frac{1}{(\mathrm{CFL} + 1\text{E-}8)^2}. \tag{13}$$

And we see that the closer CFL is to zero, the bigger the derivative in equation (13) becomes. Resulting that the network learns to stay away from CFL values near zero.

So the derivative of (11) has a larger influence when the CFL number predictions are moving away from 0 and (13) has a large influence in the beginning, when the predictions are near. The constant 1E-5 is added so that the training progress won't be influenced later on in training, when the predictions are not near zero anymore. Because the predictions of the CFL number could be relatively small for certain elements, and this prediction should not be influenced by the term in (13).

7 Results

Two different networks and two different data structures are discussed in this report. For each of the data sets we will investigate which one provides the best information for network training. For the two networks, we will discuss their performance and compare it with the performance of the default settings and with the performance of an optimized CFL number. The performance will be determined by how much iterations it takes to reach convergence.

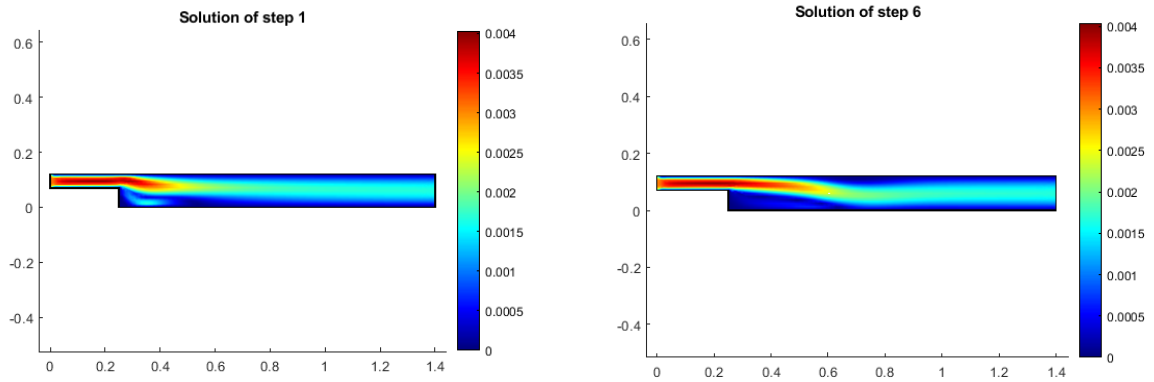
7.1 Default settings and optimal CFL

In section 5.2.2 we discussed how the optimal CFL number can be computed within COMSOL. It is of course interesting to see how much better this optimized CFL number performs compared to the two default CFL numbers discussed in section 4.2. The performance of both the default CFL numbers and the optimal CFL number will also be used to determine the performance of the trained neural networks.

First we will discuss the default settings. We start with a single normal Newton constant step, and after that we will compute each step with pseudo time-stepping turned on. The pseudo time-stepping will be turned on at the fully coupled solver, meaning that the CFL number is computed as in equation (8).

The results of the computation with the default CFL and Newton constant are shown in Figure 11. After one step, shown in Figure 11a, we can see that the solution has not converged yet. But after six steps, shown in Figure 11b, the solution won't change any more and has converged.

With the default solver automatic Newton, the convergence would take 37 iterations. Note that without pseudo time-stepping and automatic Newton, the solution will not converge at all. This shows again the importance of solver and solver setting choice.



(a) Solution after one step

(b) Solution after six steps

Figure 11: Solution of the back-step simulation after one and six steps with pseudo time-stepping with default CFL and Newton Constant.

To determine the performance of an optimized CFL number the optimization study and a single Newton step has to be computed simultaneously. First we start with a single, normal Newton step without pseudo time-stepping. The solution will be used as an input for the optimization study and as an initial value for the next Newton step with pseudo time-step that uses the optimized CFL number. In Figure 12 we can see that already after one step the solution has almost converged. Comparing this result with the default CFL, we can see that having an available optimal CFL number result in faster convergence of the solution.

The optimal CFL numbers were now computed with an optimization step in COMSOL, which took around 20 minutes to optimize the CFL numbers for one Newton step. Performing this optimization is time-wise too expensive to use on a regular or standard basis. However, if a neural network can predict the optimal CFL number, the pseudo time-step algorithm could be able to give a converged solution with less Newton steps but without extra computational time.

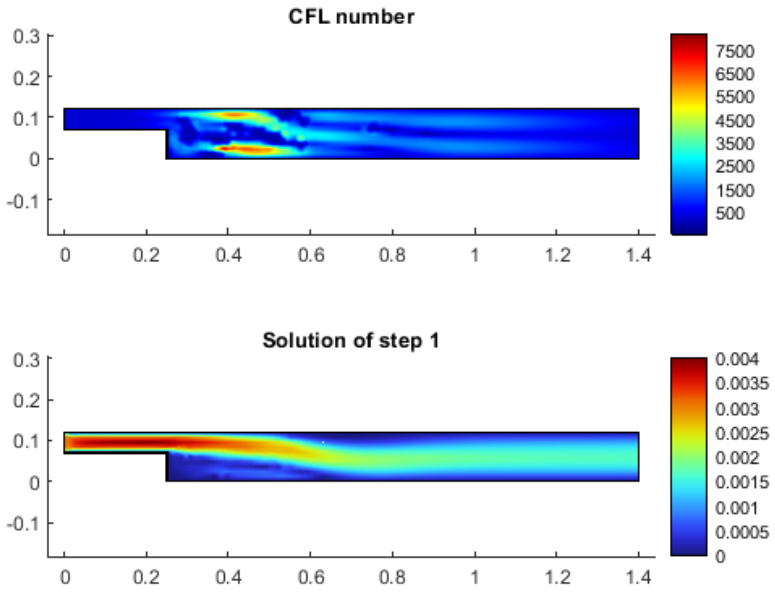


Figure 12: back-step solution shown below after one step with optimized CFL number shown above.

7.2 Network with optimized CFL target

The network discussed in section 6.2 uses the optimized CFL number as a target for the learning process. We will discuss the learning process of the network, as well as the output of the network compared to the default settings and the optimal CFL number.

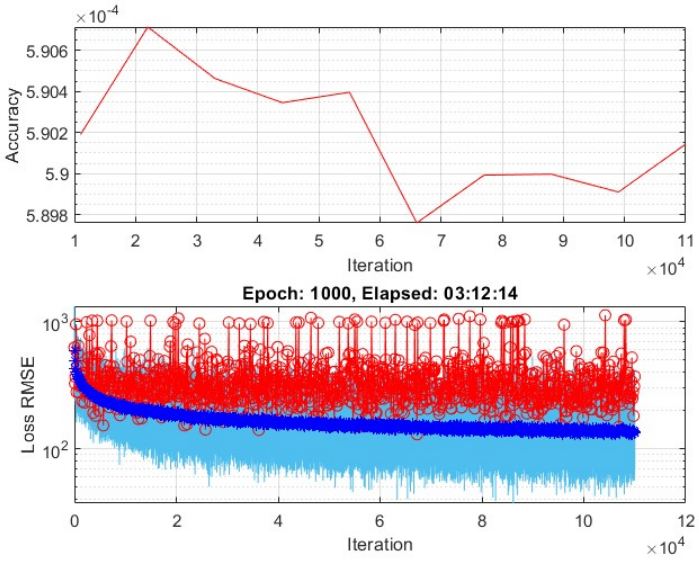
To also measure how effective the predicted CFL number is, an accuracy will also be measured during the training of the network. This accuracy will be equal to the loss of the second network, shown in equation (11). The training of the network is performed on both of the data sets, which we will discuss now and the results will be compared.

Ultimately, the effectiveness of the network predictions will be determined by using the predictions as an input for COMSOL. With this inputted CFL one iteration will be computed in COMSOL, then the network will again make a prediction for the CFL for the next iteration, and this goes on until the solution has converged.

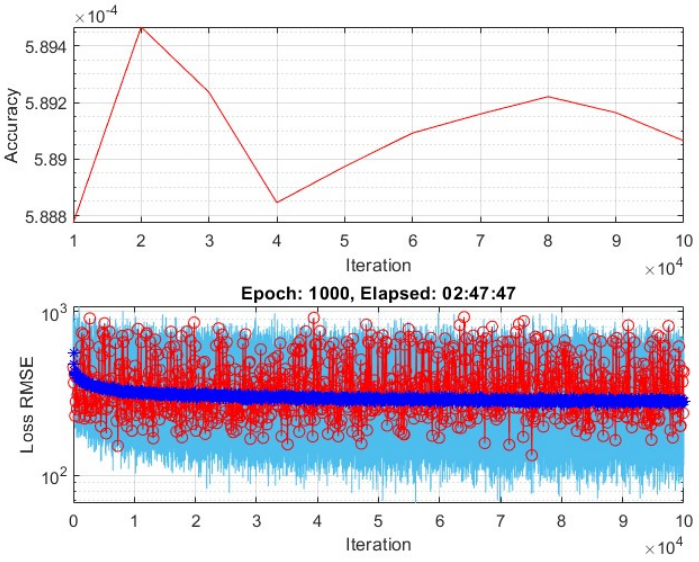
Due to some unforeseen difficulties regarding COMSOL, it has not been possible yet to use the neural network effectively. We will discuss this in more detail in section 8. In this section we will focus on the effect of the two different data sets on the training of the network, and compare their results with the target optimal CFL.

In Figure 13, we can see the training plots of the network which uses patched data in 13a and of the network which uses single element data in 13b. The accuracy plot shows the computed accuracy of the network, which is equal to equation (11). A smaller value would mean a more accurate network. For both the patched data and the single element data, the value of the accuracy is around the same.

If we look at the loss plot however, we can see come differences between the two data sets. The network that is trained with patch data seems to minimize the loss faster than the network that uses data from a single element. The loss computed with the network that uses a single element does not seem to decrease a lot after the first couple of epochs. That means that the extra information does lead to better learning performance regarding the minimization of the loss. This also leads to overfitting, as we can see in Figure 13a that the validation loss in red stays above the mean loss in dark blue. In Figure 13b we see that the values of the validation loss is still around the mean loss, so the network is not yet overfitting.



(a) Training of the network with patched data



(b) Training of the network with single element data

Figure 13: Training plots with the accuracy above, which is equal to equation (11), and the loss below. In the loss plot, light blue is the loss per iteration, dark blue is the average loss per epoch and red is the validated loss per epoch. The loss is the RMSE between the output and the target CFL number.

Besides the training plots, the predicted CFL numbers of the networks and the optimized target CFL are shown in Figure 14. The validation data is one entirely separate Newton step from the training and testing data. This is because all the elements of one Newton step must be used to predict a CFL number. Separating the validation data and the test data prevents that the computation of the validation loss and the plot of the predicted CFL are used for training.

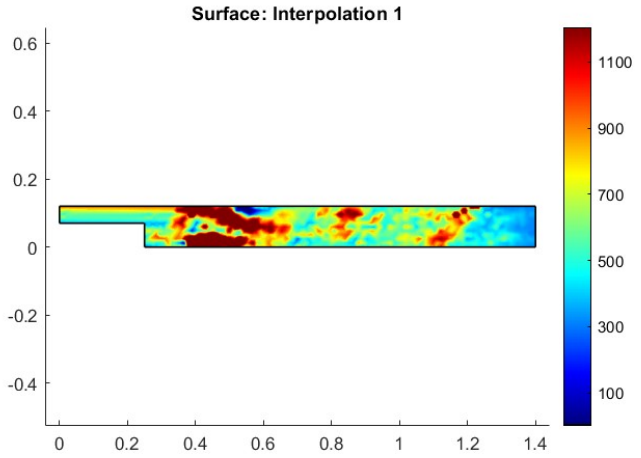
In Figure 14c we have the optimized CFL target. This is the CFL number that the network should be able to predict. The prediction of the network that uses patched data is shown in Figure 14a and of the network that uses single element data is shown in Figure 14b.

We see that both predictions have more large-valued CFL numbers than the reference CFL number, not only locally but also globally. The predictions with the patch data is predicting higher CFL number than the predictions with the single element data. Although the CFL number of the predictions are a bit higher than the target and the structure is also not exactly the same, it is not completely random. On the places where larger CFL numbers are expected, the networks also predict larger CFL numbers.

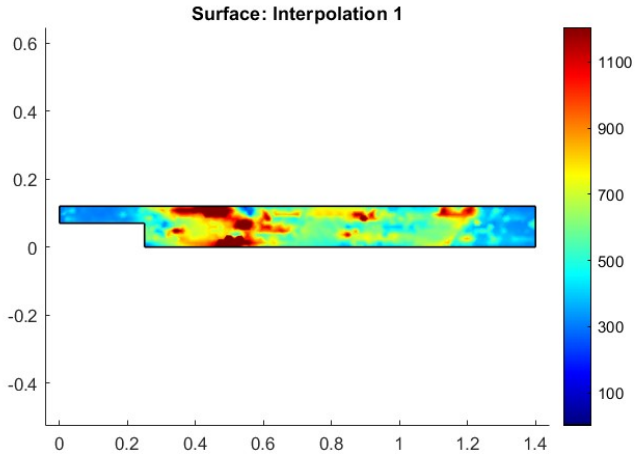
Of course, these pictures do not show how well the actual prediction is. Ideally we want to use the network to make prediction for the CFL for a full simulation, in order to see if it performs better than the 6 steps required for convergence of the default settings. Further research in section 8 will explain more about this.

7.3 Network with objective to minimize residuals

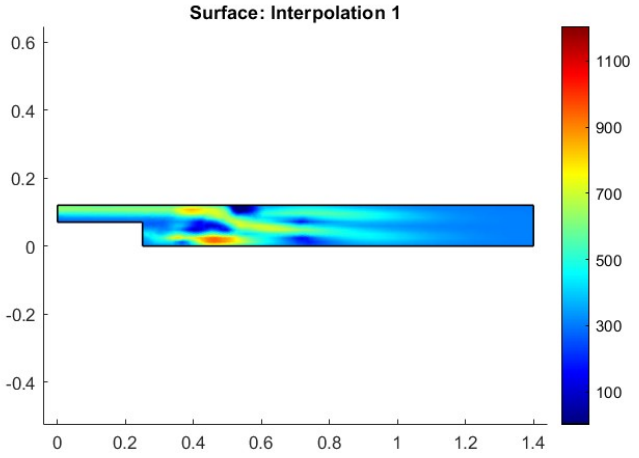
This network that was discussed in section 6.3 uses COMSOL to compute both a part of the loss and a part of the gradient. The initial idea of the research was to create a neural network that is generalizable and can use data from other models directly. This network is able to do that, since it uses local data and the loss is computed directly with the given model. However, creating such a network such that it can make correct predictions is not straight forward. In the literature, it was shown in [6] and [7] that it is more common to use known target values to train a network.



(a) Predicted CFL number with patch data set



(b) Predicted CFL number with single element data set



(c) Reference optimized CFL number

Figure 14: Reference optimized CFL number and predicted CFL number by the neural networks using the two different data sets.

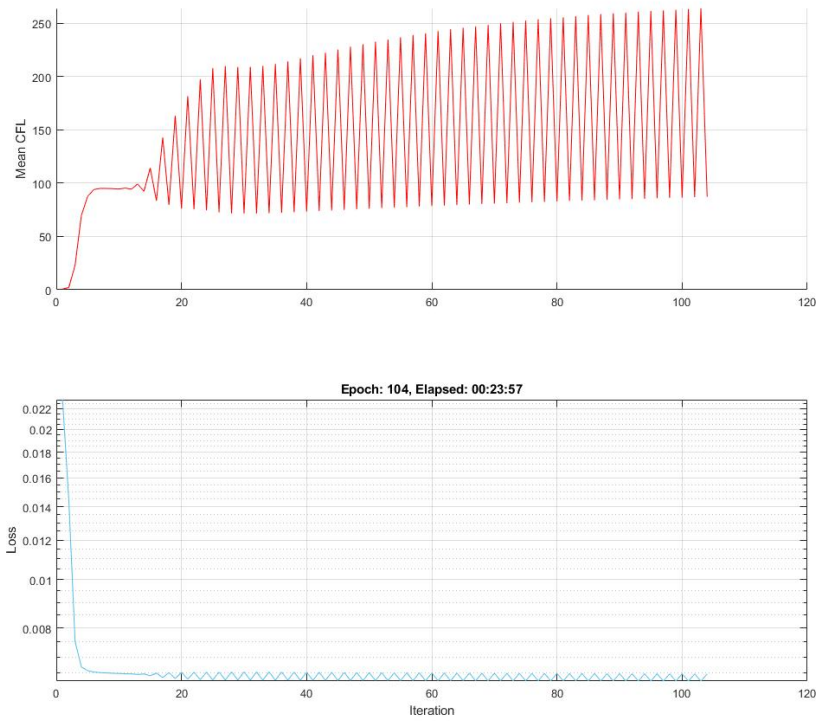


Figure 15: Training of the network with the mean CFL shown above and the loss shown below.

Since training this network brings more difficulties, there are not yet any good results to be used. As we can see in Figure 15, after some iterations during training the loss and the means CFL number are oscillating heavily. The mean CFL is given, since the output of the network is almost everywhere the same, as you can see in Figure 16.

What is also shown in Figure 16, is the fact that the CFL numbers outputted by the network does not show the structure of reference CFL number, which in this case is shown in Figure 7. The structure of the predicted CFL is like this in the first epochs, but after more and more epochs, the CFL number prediction will in fact be the same everywhere in the back-step domain. After two days of learning, the CFL number tends to go to a value between 300 or 400. That would be equal to the optimal global CFL shown in Figure 6. It could be possible that the CFL number predictions will become more diverse when this global value is reached, however this has not been investigated yet.

In conclusion: for this network to make more accurate predictions of the optimal CFL number, more work and research are required. This will be discussed in section 8.

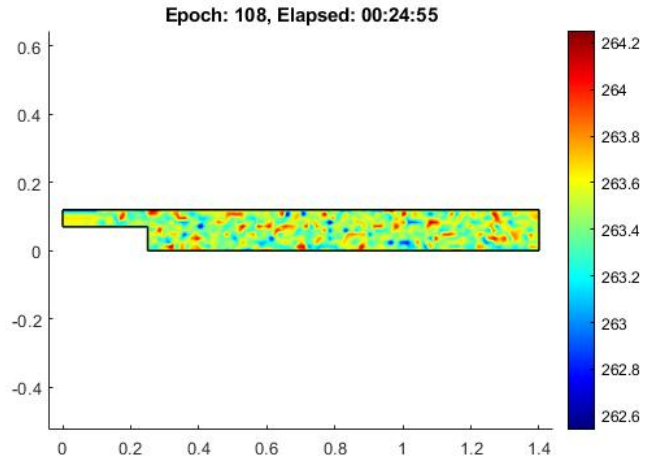


Figure 16: Predicted CFL number of the network after 108 epochs of training.

8 Further research

Two neural networks are build in Matlab and both of them require some more work and research. Each network has their own challenges to face in order to improve the outcome and to obtain correct and effective CFL predictions. But not only the networks require further research, also the data sets used for training the networks still need to be further investigated.

8.1 Networks

The network that uses an optimized CFL target for the loss, explained in section 6.2, showed that it is capable of making a prediction that is structure-wise around the same as the optimized CFL used as a target. Still, further research is required to determine how well the network performs and how well the predictions are.

The network that directly inputs the predictions to determine the loss, explained in section 6.3, is not yet capable of making a good prediction. For both networks we will discuss further research required to improve them and to determine how well they actually perform.

8.1.1 Network with optimized CFL target

As we could see in section 6.2, the network predictions are already promising. However, how well these predictions are and how well they perform are still unknown. This will be one of the first thing to investigate. In order to investigate this, COMSOL first has to perform one Newton iteration step. The data of this step will be used as an input for the network that then can make a prediction for the CFL number. This CFL number is putted back in into COMSOL and the next Newton step with this CFL number will be computed. This goes on until convergence.

For the default pseudo time-step settings and the optimal CFL number we have an indication of the amount of iterations required to obtain a converged solution, discussed in section 7.1. The aim for the network is to reach convergence with the same amount or, preferably, less iterations than the default pseudo time-step settings.

In addition, there has not been performed a grid search on the neural network yet. It still has to be investigated how many hidden layers and neurons per hidden layers result in the best performing network. Also, a validation patience can be added to further improve the results of the network and make performing the grid search faster.

Lastly, since local data is used to create a generalizable network, it is therefore also important to investigate the generalizablility of the network. If it can perform well for the back-step model, other laminar flow model can be created so that the network can be applied on them. Then it would also be interesting to see if the network predictions improve the default settings, perform equally well as the default settings, or that it performs worse.

8.1.2 Network with objective to minimize residuals

This network, explained in section 6.3, still requires more investigations in order to output a reasonable CFL number. As we can see in Figure 16, the predicted CFL numbers do not have a certain structure, it does not look like the reference CFL shown in Figure 8, and the outputted CFL numbers are almost all the same.

Since the concept of directly using COMSOL to compute the loss and train the network is interesting for generalizability and ability to apply and re-train on different fluid flow models, the investigation on this network will still continue. There are several possibilities to improve this network: using larger patches, improve the data sets, train for a longer amount of time or use a recurrent neural network. The last option will be discussed in section **??**.

When the predictions of the network improve, we can also investigate the performance of the network like explained in section . A grid search can then also be performed, as well as adding a validation patience. But before this will be research, the main focus will now be on the research of the network in section due to the limited amount of time.

8.2 Data sets

In section 6.1 the data sets are thoroughly discussed. There are currently two data sets: one that consists of information of only one element and one that consist of information of a patch of four elements. Each element gives the same information: velocities and residuals on the vertices, the Reynolds number in the centroid, and the edge lengths.

These data sets seem to perform well for training the network with the optimized CFL target. However, using these data sets in practice result in some problems.

Firstly, the iteration count is discussed. When creating a data set, data points from several Newton iterations are included in the data set. Normalizing the data set for training the network then doesn't bring any problems. However, if the network is used in practice, the data set only contains information of one Newton iteration. That means the iteration count is the same for every data point, and thus normalizing this data is not possible. Besides this, the iteration count also might give misleading information, since one flow model can converge with less Newton iterations than some other flow model. Therefore, for further research, the iteration count will not be included in the data sets anymore.

Secondly, one of the residuals used, `residual(u)`, actually takes the residual of the previous Newton iteration. Resulting in this residual being zero after the first iteration. It has to be figured out if it is either better for the network to not include this residual, or if the information of the residual has to be shifted by one iteration. The drawback of the first option, to not include it, is that possible necessary information for the network is being missed. The drawback of the second option, shift the information, is that it is more computationally expensive. However, since it is already difficult and new to create a network that can predict solver parameters, including and shifting the residuals would be the better option now. If the network turns out to perform well with this information, one can always investigate how to make the network performing more efficient.

Besides improving the input data itself, the patched data can also be improved. At this point, only elements with four neighbouring elements are included in the patched data set. This means that boundary elements are not included in the data sets. Therefore, CFL predictions are extrapolated to the boundary. In order to improve the network training and predictions, it is necessary to also include the boundary elements in the data set. This can be relatively easily achieved by setting the information of the missing edges and vertices on zero.

Next to including boundary element data, investigating if the data can be structured so that there is no overlapping data in a data point can also improve the training by making the data set less complex.

Now, information of vertices and edges that have respectively three or two elements in common is also in the data point three and two times. Removing this redundant information by structuring the data can also improve the training and performance of the neural network.

References

[1] Peter Gainsford. *COMSOL Multiphysics Reference Manual*. v. 5.6. COMSOL. Stokholm, Sweden, 2020.

[2] Aurélien Géron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. 1st ed. Sebastopol, CA: O'Reilly Media, Mar. 2017. ISBN: 9781491962268.

[3] Carl Kelley and David Leyes. "Convergence Analysis of Pseudo-Transient Continuation". In: *SIAM Journal on Numerical Analysis* 35 (Aug. 1996). DOI: 10.1137/S0036142996304796.

[4] Randall J. LeVeque. *Finite Volume Methods for Hyperbolic Problems*. Cambridge Texts in Applied Mathematics. Cambridge University Press, 2002. DOI: 10.1017/CBO9780511791253.

[5] *LiveLink for MATLAB User's Guide*. v. 6.0. COMSOL. Stokholm, Sweden, 2021.

[6] Nils Margenberg et al. "A neural network multigrid solver for the Navier-Stokes equations". In: *Journal of Computational Physics* (Jan. 2022), p. 110983. ISSN: 0021-9991. DOI: 10.1016/j.jcp.2022.110983. URL: http://dx.doi.org/10.1016/j.jcp.2022.110983.

[7] Deep Ray and Jan S. Hesthaven. "Detecting troubled-cells on two-dimensional unstructured grids using a neural network". In: *Journal of Computational Physics* 397 (2019), p. 108845. ISSN: 0021-9991. DOI: https://doi.org/10.1016/j.jcp.2019.07.043. URL: https://www.sciencedirect.com/science/article/pii/S0021999119305297.

[8] *Turbulent flow over a backward-facing step*. URL: https://www.comsol.com/model/turbulent-flow-over-a-backward-facing-step-228 (visited on 03/17/2022).

[9] J. Watt, R. Borhani, and A. Katsaggelos. *Machine Learning Refined: Foundations, Algorithms and Applications*. Cambridge: Cambridge University Press, 2016.

[10] P. Wesseling. *Elements of Computational Fluid Dynamics*. Delft: TU Delft, 2014.

Appendix

A Interface between COMSOL and Matlab

LiveLink for Matlab introduces a connection from COMSOL to the Matlab environment [5]. Within this LiveLink, it is possible to set up COMSOL models from Matlab, use Matlab functions in COMSOL, set up model loops and of course analyze the results of the COMSOL simulation [5].

For this thesis, the interface was used to obtain COMSOL simulation results, create data sets and use the neural network output as an input for the COMSOL simulation. First the model is loaded with `mphload`.

Obtain COMSOL simulation results

In order to use data from the COMSOL simulation, you first have to obtain that information in Matlab. There are several functions to obtain result data, such as flow velocities, pressure and the residuals. But also the results of the sensitivity analysis should be obtained, in order to compute the gradients in section 6.3.3.

There are two methods that are used in this research and both will be discussed. These methods are `mpheval` and `mphgetu`. We will first discuss `mpheval`.

The function `mpheval` is used to obtain solutions and defined variables from the COMSOL simulation. The variables obtained are u, v, p and their residuals and the Reynolds number. The code to obtain the solution vectors is given as follows:

```
outcome = mpheval(model,{'u','v','p'},'refine',1,'dataset','dset10');
```

Besides the values you want to obtain from the simulation, it is also possible to state from which data set the solutions must be obtained. `outcome` also gives information of the coordinates of each point in the solution vector.

Secondly, `mphgetu` is used to obtain the solutions of the sensitivity analysis. A drawback of this function it that it returns one large vector instead of multiple vectors that represent each of the three variables of the sensitivity analysis. That means you need to separate the vector afterwards. Luckily, the vector is ordered logically and it is not a big deal to separate the vector. The code is given as follows:

```
outcome2 = mphgetu(model,'type','fsens','soltag','sol9');
```

Again, it is possible to state from which solution you want to obtain the results with the use of `'soltag'`. Also, `outcome2` does not have any information of the coordinates of each solution point in the vector. In order to obtain the right coordinates, you must use `mphxmeshinfo`:

```
info = mphxmeshinfo(model);
coords = info.nodes.coords;
```

These coordinates are not ordered the same as the coordinates obtained with `mpheval`, therefore a permutation vector is created to effectively shift the values into the same coordinates.

Change settings in COMSOL

During the training of the neural network and data creation, settings for the COMSOL model must be adapted. During the training we want to turn the CFL value to cfl, so we have to refresh the interpolation function, activate manual CFL value and set it to cfl.

```
model.func('int1').active(true);
model.component('comp1').func('int1').refresh;
model.physics('spf').prop('AdvancedSettingProperty').set('CFLNumbExpr',
 ↪  'Manual');
model.physics('spf').prop('AdvancedSettingProperty').set('locCFL', 'cfl');
```

When a data set is created, the CFL number is set to manual and the maximum number of iterations is set to a certain value n.

```
model.physics('spf').prop('AdvancedSettingProperty').set('CFLNumbExpr',
↪  'Automatic');
model.sol('sol8').feature('s1').feature('fc1').set('niter', num2str(j));
```

For finding out which Matlab operation should be used, the COMSOL Model Navigator app is used. This app navigates through the loaded COMSOL simulation.

Create data sets

As mentioned in section 6.1, there are 22 network input values for the part of the network that predicts the CFL number, and there are 2 network inputs that pass information of the coordinates to the regression output layer. In order to obtain this information, the first study step should be run. But first, general information independently of the solution is determined, such as the element edge lengths.

```
model = mphload('backstep.mph');

info = mphxmeshinfo(model);
coords = info.nodes.coords;

for i = 1:size(info.elements.tri.nodes,2)
    edge_length1(i) = norm( [coords(1,mesh_info(2,i))
↪  coords(2,mesh_info(2,i))]-[coords(1,mesh_info(3,i))
↪  coords(2,mesh_info(3,i))] );
    edge_length2(i) = norm( [coords(1,mesh_info(2,i))
↪  coords(2,mesh_info(2,i))]-[coords(1,mesh_info(4,i))
↪  coords(2,mesh_info(4,i))] );
    edge_length3(i) = norm( [coords(1,mesh_info(4,i))
↪  coords(2,mesh_info(4,i))]-[coords(1,mesh_info(3,i))
↪  coords(2,mesh_info(3,i))] );
end
```

Then, for a given number of iteration n, the first study step is run. The results of this study step must be obtained for both the element vertices and the element centroids. The residuals are only required for the element vertices.

```
model.sol('sol8').feature('s1').feature('fc1').set('damp', '1');
model.physics('spf').prop('AdvancedSettingProperty').set('CFLNumbExpr',
↪  'Automatic');
model.sol('sol8').feature('s1').feature('fc1').set('niter', num2str(n));
model.sol('sol8').runAll;

data = mpheval(model,{'u','v','p','spf.res_u','spf.res_v',...
    'spf.res_p','spf.cellRe'},'refine',1,'dataset','dset10');
```

All the required data for the solution are put into a table and this table is then written to an Excel file.

```
writetable(data_table,'LocalData.xlsx','Sheet',n);
```

B Code structure neural network

The code of the neural network consists of several part with each their own function. There is of course first a main file, where the information necessary prior to the network training is given. Then a function is called where the data sets for training, validation and testing are created, as well as the neural network and the mini-batch. In this same environment the network is trained. When the network is created, the output regression layer is also created. This is a custom layer with custom loss and gradient, which both also have their own function. We will walk through every aspect of the code, starting with the main file.

Main

In the main file, the user can define the parameters such as number of layers, number of iteration and the filenames for the data set and COMSOL file. With the use of the COMSOL file, a permutation matrix is created to be able to match the coordinates of the `mphgetu` and `mpheval`. This information is used as an input of the custom function `TrainNeuralNet`. In this file we have a custom function that creates data sets, creates the neural network and trains the neural network.

Create data sets

First in the this file, the function `give_data` gives the array data stores and tables for the whole data set, the training data set, the validation data set and the test data set. With these array data stores, the different mini-batches are created.

The code for generating the table and array data store for the training set is given below.

```
tbl = readtable(filename,'Sheet',iter,'TextType','String');
numObservations = size(tbl,1);
numObsTrain = floor(0.7*numObservations);

idxTrain = idx(1:numObsTrain);
tblTrain = tbl(idxTrain,:);
ds = combine(arrayDatastore(table2array(tblTrain(:,1:22))', ...
    IterationDimension = 2), ...
    arrayDatastore(table2array(tblTrain(:,23:24))', ...
    IterationDimension = 2),...
    arrayDatastore([zeros(1,size(tblTrain,1)); ...
    table2array(tblTrain(:,23:24))'],IterationDimension = 2));
```

Create the neural network

Secondly, the neural network is build with the given number of layers, neurons and mini-batch size. As shown in figure 10, there are two input layers, several hidden layers, a concatenation layer and finally a custom defined regression output layer. The code for creating the neural network can be found in appendix section C.

Create custom regression output layer

Within this layer, both the gradient and the loss of the network are computed. The only property of the custom regression layer is the table of the complete data set, which is used for the computation of the gradient. First the gradient is computed, secondly the loss is computed. In section 6.3.3 it is explained how the gradients are computed and in section 6.3.2 it is explained how the loss is computed. But besides the computation of both the gradient and loss, it is important to obtain the predicted CFL number and put them back in COMSOL with the right coordinates.

In order to put the CFL numbers in COMSOL, a text document is used. In this text document, all the CFL numbers with corresponding coordinates are written. Since the network first computes the gradient, the text file is adapted in the file where the gradient is computed.

With this text file, the interpolation function that uses this information is activated and refreshed. Then the model is run to obtain the required solution vectors, residuals and sensitivity analysis outcomes to compute the gradient.

```
model.func('int1').active(true);
model.component('comp1').func('int1').refresh;
model.sol('sol9').runAll;
```

When the loss is computed, it can directly use the text file and compute the loss with the obtained residuals from the COMSOL simulation.

Train neural network

For training of the neural network the function `trainNetwork` is used.

C Matlab code

Create neural network

```
layer = layerGraph;
NumNeurons = Neurons;
NumLayers  = Layers;
    reluname = '';
    layer1 = [
        featureInputLayer(22,'Normalization','none','Name','input')
        batchNormalizationLayer('Name','BN1')
        fullyConnectedLayer(NumNeurons,'Name','fc_begin')
        reluLayer('Name','Relu1')];%activation function
    layer = addLayers(layer,layer1);
    for i = 1:NumLayers
        reluname2 = reluname;
        reluname = append('Relu',num2str(i+1));
        layername = append('fc_',num2str(i));
        layer2 = [fullyConnectedLayer(NumNeurons,'Name',layername)
            reluLayer('Name' ,reluname)];
        layer = addLayers(layer,layer2);
        if i>1
            layer = connectLayers(layer, reluname2, layername);
        else
            layer = connectLayers(layer, 'Relu1', layername);
        end

    end


    layer5  = [fullyConnectedLayer(1,'Name','reg_end')
        softplusLayer('Name' ,'lastsigmoid')];
    input2 = featureInputLayer(2,'Normalization','none','Name','input2');
    concat  =    concatenationLayer(1,2,'Name','concat');
    reglayer=    myRegressionLayer2(tbl,2,miniBatchSize,perm_mat,'last');
    layer = addLayers(layer,layer5);
    layer = addLayers(layer,concat);
    layer = addLayers(layer,reglayer);
    layer = addLayers(layer,input2);
    layer = connectLayers(layer,reluname,'reg_end');
    layer = connectLayers(layer,'lastsigmoid','concat/in1');
    layer = connectLayers(layer,'input2','concat/in2');
    layer = connectLayers(layer,'concat','last');
```