Neural Network-based Optimization of Solid-and Fluid Mechanical Simulations

PABLO JEKEN RICO

Double Degree Master with TU Delft in Computer Simulations for Science and Engineering
Date: 31/08/2021
Supervisor: Matthias Wohlmuth
Examiner: Michael Hanke
School of Electrical Engineering and Computer Science
Host company: BASF SE
Programme Coordinator at TU Delft: Kees Vuik

Abstract

The following project deals with the optimization of simulation parameters such as the injection location and pitch angle of polyurethane foaming simulations using artificial neural networks. The model's target is to predict quality variables based on the process parameters and the geometry features. Through several evaluations of the model, good parameter combinations can be found which in turn can be used as good initial guesses by high fidelity optimization tools. For handling different mould geometries, a meshing tool has been programmed which transforms variable-sized surface meshes into voxel meshes. Cross-section images of the meshes are then passed together with a series of simulation settings to the neural network which processes the data streams into one set of predictions. The model has been implemented using the TensorFlow interface and trained with a custom generated data set of roughly 10000 samples. The results show well-matching prediction and simulation profiles for the validation cases. The magnitudes of the quality parameters often differ, but the especially relevant areas of optimal injection points are well covered. Good results together with a small model size provide evidence for a feasible and successful extension towards a full 3D application.

Keywords

PU Foaming Simulations, Computer Vision, Artificial Neural Networks

Sammanfattning

Neurala nätverksbaserad optimering av mekaniska simuleringar av fasta och flytande ämnen

Följande projekt handlar om optimering av simuleringsparametrar, såsom injektionsplats och stigningsvinkel för polyuretanskummande simuleringar med hjälp av artificiella neurala nätverk. Modellens mål är att förutsäga kvalitetsvariabler baserat på processparametrarna och geometrifunktionerna. Genom flera utvärderingar av modellen kan man hitta goda parameterkombinationer som i sin tur kan användas som gedigna förutsägelser med högkvalitativa optimeringsverktyg. För hantering av olika geometriska former har ett maskverktyg programmerats som omvandlar ytmaskor med varierande storlek till voxelmaskor. Tvärsnittsbilder av maskor na tillsammans med en serie simuleringsinställningar överförs till det neurala nätverket som behandlar dataströmmarna till en uppsättning förutsägelser. Modellen har implementerats med hjälp av TensorFlow och utbildats med en anpassad genererad datauppsättning på cirka 10000 prover. Resultaten påvisar väl matchande förutsägelser och simuleringsprofiler för valideringsfall. Kvalitetsparametrarnas storlek varierar ofta, men de särskilt relevanta områdena med optimala injektionspunkter är väl täckta. Goda resultat tillsammans med en liten modellstorlek ger bevis för en genomförbar och framgångsrik förlängning mot en fullständig 3D applikation.

Nyckelord

Polyuretanskumning Simuleringar, Computer Vision, Artificiella Neurala Nätverk

Acknowledgements

This master thesis has been done to conclude the COSSE master programme hosted by the KTH Stockholm (KTH), TU Delft (TUD) and TU Berlin. The research has been supported and sponsored by the Ultrasim Department of BASF based in Ludwigshafen am Rhein. The degree project started in February 2021 and was concluded in August 2021 in Stockholm, Sweden.

BASF SE
Carl-Bosch-Straße 38
67063 Ludwigshafen am Rhein

First of all, I would like to thank my supervisors Matthias Wohlmuth (BASF) and Michael Hanke (KTH) who, despite the covid-related odds, have provided me with continuous and supportive guidance in this project. Further, I would like to thank my work colleagues Bruno Schiebler (BASF) and Patrick Baumgart (BASF) for their extensive help with the IT-related work. I have learned plenty about proper Python programming and the Linux environment thanks to them. Finally, I would like to thank my fellow students Roman Delorme (RWTH) and Markus Peschl (TUD) for providing me with valuable suggestions for the model assessment and validation.

Contents

List of Figures

List of Tables

Acronyms

A Air.

AI Artificial Intelligence.

ANN Artificial Neural Network.

BA Blowing Agent.

BGD Batch Gradient Descend method.

CFD Computational Fluid Dynamics.

DNNs Deep Neural Networks.

F Polyurethane Foam.

GD Gradient Descend method.

ISO Isocyanate.

MAE Mean Absolute Error.

ML Machine Learning.

MLP Multi-Layer Perceptron.

MSE Mean Squared Error.

OH Polyol.

PDEs Partial Differential Equations.

PU Polyurethane.

SGD Stochastic Gradient Descend method.

W Water.

Symbol Directory

Symbol	Dimension/Size	Description
α	$dimensionless$	Alpha phase.
β	$[K^{-1}]$	Isobaric volume expansion.
c	$[mol/m^3]$	Chemical concentration.
c_p	$[J/K]$	Isobaric heat capacity.
d	$integer$	Sample index.
$\hat{d}$	$dimensionless$	Dropout rate.
$\mathcal{D}$	$n_{samples}$	Set of samples.
$\mathbf{D}$	$[s^{-1}]$	Rate of deformation tensor.
E	$[J]$	Energy.
$\hat{E}$	$[J/mol]$	Activation energy.
η	$dimensionless$	Stepsize in gradient based optimization.
$\hat{\eta}$	1	Momentum controlled step size.
f	$dimensionless$	Model function for derivations.
$\mathbf{f}$	$[N/m^3]$	Body forces.
$\mathbf{g}$	$[m/s^2]$	Gravity vector.
g	$dimensionless$	Activation function.
$\dot{\gamma}$	$[s^{-1}]$	Shear rate.
$\hat{H}$	$[J/kg]$	Specific enthalpy.
ΔH	$[J/mol]$	Molar reaction enthalpy.
$\mathbf{I}$	$[1]$	Identity matrix.
κ	$[J/m^2s]$	Thermal conductivity.
L	$dimensionless$	Loss/Error function.

λ	$[kg/ms]$	Bulk viscosity.
μ	$[kg/ms]$	Dynamic viscosity.
p	$[N/m^2]$	Pressure.
p_t	$[N/m^2]$	Thermodynamic pressure.
$\mathbf{q}$	$[J/m^2s]$	Heat flux.
Q	$-$	Source terms.
ρ	$[kg/m^3]$	Density.
$\hat{S}$	$[J/kgK]$	Specific entropy.
σ	$[N/m^2]$	Cauchy stress tensor.
T	$[K]$	Temperature.
$\hat{U}$	$[J/kg]$	Specific inner energy.
$\hat{V}$	$[m^3/kg]$	Specific volume.
$\mathbf{v}$	$[m/s]$	Velocity.
$\nabla^n_{\mathbf{v}}$	$dimensionless$	n^{th} directional derivative along $\mathbf{v}$.
X	$[1]$	Dimensionless chemical conversion.
$\mathbf{x}$	$n_{features}$	Feature vector.
y	–	Dependent variable.
$\mathbf{y}$	$n_{samples}$	Vector of dependent variables y.
w	$dimensionless$	Weight value.
$\mathbf{w}$	–	Weight vector.

Chapter 1

Introduction

Polyurethane (PU) foams are a widely used class of polymers that can be found all across industrial machinery, buildings, and all sorts of means of transport. There are countless variations of the basic PU, many of which aim to meet specific requirements, for instance, rigid foams used in insulation layers in thermal applications or soft foams with which sponges and acoustic damping walls are made [1].

(a) Insulation panels of cooling facility. (b) Car interior.

Figure 1.1: Applications of PU foams [2].

Objects made of foam materials are generally produced by mixing at least two liquid reactive components and distributing them over a surface or volume to obtain the desired shape. The applications that involve filling a mould with PU foam require an extensive analysis before a production line can start. It is desirable to distribute the liquid polymers in a way that the whole component has an isotropic density and strength after the foaming reaction. At this stage, Computational Fluid Dynamics (CFD) simulations of the foaming process can provide valuable information that helps engineers to assess the production plan, leading to optimized production plans using the aforementioned results. Improving the geometry and production parameters requires often several cycles of simulations, optimization and adaptations, which can lead to

extended development times. Given this issue, it would be supportive to find a method that can learn from previous simulations to suggest helpful setups for new models in a feasible time.

Artificial Neural Networks (ANNs) belong to a fast-growing branch of Machine Learning (ML) techniques and bear the potential to solve the underlying issue. Due to their capability of finding highly non-linear correlations in both categorical data and geometrical complexes they are expected to be suited for the proposed optimization goal.

The thesis project starts with an introduction to PU foam rheology and simulations (Section 2), followed by the analysis of ANNs and their optimization methods (Section 3). In Section 4, the automatized generation and processing of geometries as well as the essential simulation workflows will be presented together with the final neural network. In Section 5 the results will be shown, analyzed and discussed for further use in an extended 3D model. Lastly, the overall goals and accomplishments will be commented in the conclusion (Section 6).

Chapter 2

Foaming Simulations

2.1 PU Foams

Polyurethanes belong to the family of reaction polymers and have many different variations. Common examples of the PU family are adhesives, coatings and lastly foams, which will be the matter of subject in this thesis. PU foams can be further divided based on their morphology into one of two groups, rigid- and flexible foams. As the name suggests, one comprehends tougher, rigid foams used in thermal insulation or shock applications. Flexible foams on the other side are used in mattresses, sponges and sound damping material due to being soft and porous.

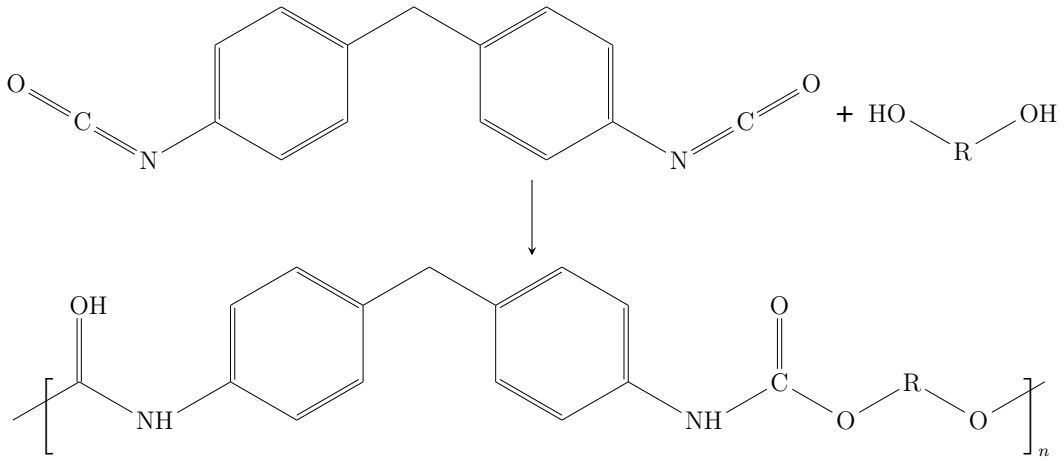

Figure 2.1: Reaction of di-isocyanite (left) and polyol (right) to polyurethane (bottom) with an organic rest group R.

PU foams arise from the reaction of a Polyol (OH) with an Isocyanate (ISO) under the presence of a catalyst and a so-called Blowing Agent (BA). In figure 2.1 a generic reaction scheme is

depicted that shows the polymerization reaction of the two initial compounds. Depending on the choice of the active group R and the isocyanate one can obtain different chain lengths and degrees of entanglement, which in turn define the mechanical and chemical properties of the product. The blowing can either be achieved by using a chemical agent that yields a gas in a secondary reaction or by a physical agent, a compound that remains inert during the foaming, but expands forming cavities in the material. The choice, quantity and combination of blowing agents define the type of foam. Chemical agents, like Water (W), cause the foam Polyurethane Foam (F) to be more flexible and porous, while physical agents tilt towards more rigid structures [3]. The inner arrangement of the foam consists of cross-threaded polymer links that contain gas-filled cavities. This grants the material the characteristic low density alongside other properties, such as good thermal insulation and shock absorption capacity. In equation 2.1 one can see the chemical reactions with some abuse of notation.

$$\begin{aligned} \text{Isocyanate} + \text{Polyol} &\longrightarrow \text{Polyurethane} \\ \text{Isocyanate} + \text{Water} &\longrightarrow \text{Carbon dioxide} + \text{Urea} \end{aligned} \tag{2.1}$$

Moulding objects of PU foam can be achieved in many ways depending on the type of product that is being treated. For the current project, the foaming process will take place in a mould that ought to be filled by the foam material. The procedure starts by injecting the mix of compounds with nozzles into a mould. After a short period, the liquid starts rising, progressively filling the cavities and pushing air out. During the process, the geometry may be also tilted to achieve a better and more homogeneous distribution of plastic. Before filling the form, the face that initially was open for the injection instruments is closed and sealed to also guide the foam into the correct shape at that part of the mould. This technique can be used to create single-piece components of vehicles and refrigerators among others.

As with most other industrial processes, getting an optimal production setup requires several iterations of optimization and testing. Factors to improve are normally shortening flow paths, avoiding trapped air as well as minimizing the density variation. Improving the quality of the results can be achieved both by cycles of physical experiments and using high fidelity CFD simulations of the foaming process. Due to the availability of computational resources, simulations have become increasingly popular and common. The foaming process is notably complex; it involves the simulation of a non-Newtonian fluid with quickly varying density in a gas-filled irregular geometry. Nevertheless, under reasonable assumptions and simplifications, a numerical approximation of the foaming may be done.

2.2 Reaction Dynamics

Foaming simulations differ from other common CFD applications by the complex reactions and transitions in the material. To start with it is important to give some insight into how the reaction dynamics are described. They in turn define the physical properties of the foam, as well as other influences like the reactions exergy. From a simplified point of view, the foaming process undergoes two stages, the injection and the foam rise. During the first stage, the polymer mixture behaves, similarly to thermosets, like a general shear-thinning fluid. The main drives for the motion are the influence of gravity and the injection speed. In the second stage, the liquid starts to transform into a less dense foam, progressively expanding in all possible directions. Material properties, such as viscosity and thermal conductivity among others undergo a rapid change during this process. To minimize the modelling error it is necessary to accurately portray the reaction dynamics in the foam. The chemical transition depends not just on the physical conditions, like the pressure p and temperature T, but also on the local concentration c of the reactants [4]. The equations presented include several new quantities presented in table 2.1. The subscript i denotes any of the chemicals: F, A, ISO, PU, W, BA or OH. A further index 0 can be found on some quantities which simply marks the initial value.

c	Chemical concentration	mol/m^3
m	Mass	kg
X	Chemical conversion	1
x	PU mass ratio	1
A	Preexponetial factor	1
$\hat{E}$	Activation energy	J/mol
R	Ideal gas constant	$J/(molK)$
M	Molar mass	$J/(molK)$

Table 2.1: Chemical variables.

The mass ratio x_i can be expressed for any of the reactants and is calculated using:

$$x_i = \frac{m_i}{m_{PU}} \tag{2.2}$$

Dimensionless chemical conversion X_i describes the converted proportion of a chemical reactant at some point of the reaction [5]. It is defined for the reactants Polyol (OH) and Water (W)

and starts at the value 0. During the reaction, the conversion factor grows up to the value of 1, signalizing a point where the entirety of the compound has reacted.

$$X_i = \frac{c_{i,0} - c_i}{c_{i,0}} \tag{2.3}$$

Even though no explicit formula for the conversion of isocyanate is given it can be inferred if X_W and X_{OH} are known and isocyanate is assumed to be abundant in comparison to the other reactants. This can be seen from examining equation 2.1. The concentrations of OH and W diminish in the course of the reaction. Their decline can be expressed using first order differential equations:

$$Q_{OH} = \frac{dX_{OH}}{dt} = A_{OH} \exp\left(\frac{-\hat{E}_{OH}}{RT}\right)(1 - X_{OH})\left(\frac{c_{ISO,0}}{c_{OH,0}} - \frac{2c_{W,0}}{c_{OH,0}}X_W - X_{OH}\right) \tag{2.4}$$

$$Q_W = \frac{dX_W}{dt} = A_W \exp\left(\frac{-\hat{E}_W}{RT}\right)(1 - X_W) \tag{2.5}$$

The first equation is attributed to the gelling reaction in which urethane is created. The second one deals with the blowing reaction of the chemical BA. Both equations can be interpreted similarly: The reactants are progressively converted during the reaction into the reaction products. Therefore, the converted concentration will grow and remain positive throughout time yielding a positive first derivative. The effects that speed up the reaction are high temperatures and a high initial concentration of isocyanate. The depletion of reactants (OH and W) on the other side slow down the reaction. In the global scheme of equations, these ODE's will be used to model the transition of compounds as well as the foam expansion process.

The conversion of water for instance is tightly bound to the foams density evolution due to its direct relationship with the CO_2 production. The carbon dioxide finds itself embedded in the polymer mass and causes it to expand and rise. For recipes, the density can drop by a factor of 60 in a matter of a minute. The evolution of the foam's density ρ_F can be modelled as followed according to [5]:

$$\rho_F = \frac{1 + x_{W,0}}{X_W \frac{RTx_{W,0}}{pM_W} + (1 - X_W)\frac{x_{W,0}}{\rho_W} + \rho_{PU}^{-1}} \tag{2.6}$$

As will be seen later in section 2.3, the density equation will provide the material law that closes the set of equations. It remains however challenging from a numerical perspective, to accurately solve a system with drastically varying densities. Other material properties, e.g. the viscosity or thermal conductivity are difficult to model due to strong temperature variations and the chemical transition foam [6].

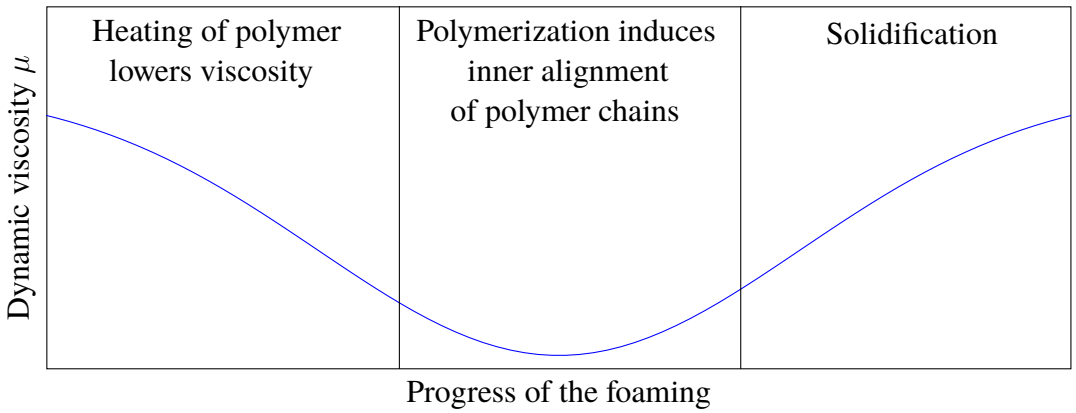

Figure 2.2: Schematic viscosity transition of thermosets abstracted from [6].

Take the case of viscosity for instance. The equations used to describe it build on known models such as the Cross-Power-Law or the Carreau model and integrate the effects of polymerization and temperature oscillations using arrays of data fitted coefficients. Despite an extensive study of literature, it remains unclear which of the models fits the physical conditions best. The same inconveniences hold for many other material properties. For this reason, they will be assumed to be known without providing their extended forms.

2.3 Balance Equations

Numerical simulations of the foaming rheology are base on systems of Partial Differential Equations (PDEs), the laws of conservation (mass, momentum and energy). The polymer foam is here assumed to be an isotropic fluid of variable density. The small bubbles of gas contained in it are not simulated since they would cause a major computation overhead without yielding any particular advantage [4]. Foam and the mould's air are therefore the two phases to consider in the equations. The following derivation will start by deriving the laws of conservation in a classical fashion for the polymer phase. The later introduced Volume-of-Fluid (VoF) method will provide the solution to the two-phase flow.

Proper modelling of fluid mechanical processes requires a set of state variables and material properties. Apart from commonly used measures for pressure p, density ρ and velocity $\mathbf{v}$ a series of state intensive variables will be needed:

p	Pressure	N/m^2
$\mathbf{v}$	Velocity	m/s
ρ	Density	kg/m^3
T	Temperature	K
$\hat{H}$	Specific enthalpy	J/kg
ΔH_i	Molar reaction enthalpy	J/mol
$\hat{U}$	Specific energy	J/kg
$\hat{S}$	Specific entropy	$J/(kgK)$
$\hat{V}$	Specific volume	m^3/kg

Table 2.2: State variables.

The conservation equations are defined over an arbitrary three dimensional domain Ω. Each of the conserved quantities $F(\mathbf{x}, t)$ equals the sum of the source and drain fluxes as well as diffusion $G(F, \mathbf{x}, t)$ in and out of the domain.

$$\frac{d}{dt}\int_{\Omega} F dV = \int_{\Omega} G dV \tag{2.7}$$

The Reynolds-Transport-Theorem is then applied to the generic equation (2.7), allowing to simplify the total derivative of a volume integral over a time dependent domain $\Omega(t)$.

$$\frac{d}{dt}\int_{\Omega} F\, dV = \int_{\Omega} \frac{\partial F}{\partial t}\, dV + \oint_{\partial\Omega} F(\mathbf{n}\cdot\mathbf{v})\, dA \tag{2.8}$$

If the surface of Ω is now assumed to be closed and piecewise smooth, the divergence theorem can be applied on the continuously differentiable vectorfield $\mathbf{v}F$, leading to:

$$\int_{\Omega} \frac{\partial F}{\partial t} dV + \int_{\Omega} \nabla\cdot(\mathbf{v}F) dV = \int_{\Omega} G dV \tag{2.9}$$

If these laws are assumed to stay true for Ω, so they have to on all of its infinitesimal subdomains, forcing the equation to be valid for the differential terms as well.

$$\frac{\partial F}{\partial t} + \nabla\cdot(\mathbf{v}F) = G \tag{2.10}$$

Following the principle of (2.10) with $F := \rho$ and $G := 0$ leads to the so called continuity equation:

$$\frac{\partial \rho}{\partial t} + \nabla\cdot(\rho\mathbf{v}) = 0 \tag{2.11}$$

The Conservation of Mass states that the total change of mass inside a closed continuum has to be equal to zero. Up next is the momentum, $F := \rho\mathbf{v}$, which leads to the second conservation law. Analogous, it can be derived as:

$$\frac{\partial \rho\mathbf{v}}{\partial t} + \nabla\cdot(\rho\mathbf{v}\mathbf{v}) = \mathbf{f_i} + \mathbf{f_o} \tag{2.12}$$

This law states that the total change of momentum of a continuum is equal to the sum of volume specific forces applied to it. The forces can be split into two types:

Internal forces

- Direct interaction with all particles inside the continuum
- Represented by the volume specific gravitational force $\rho\mathbf{g}$

External forces

- Affects the surface particles
- Responsible for a substantial part of the particle interactions
- Represented by the divergence of a fluid dependent stress tensor σ

$$\frac{\partial \rho \mathbf{v}}{\partial t} + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) = \rho \mathbf{g} + \nabla \cdot \sigma \tag{2.13}$$

The equation can be presented in a shorter form by performing the product rule on the spacial and temporal derivations and recalling (2.11).

$$\rho \frac{\partial \mathbf{v}}{\partial t} + \mathbf{v} \left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) \right) + \rho \mathbf{v} \cdot \nabla \mathbf{v} = \rho \mathbf{g} + \nabla \cdot \sigma \tag{2.14}$$

It remains now to analyze the implications of the stress tensor σ. Polymers as the ones used for PU foaming can be assumed to be Generalized Newtonian Fluids in their liquid state. This assumption is accurate enough, even though their properties change when transitioning into a foam [4]. The stress tensor σ is given by:

$$\sigma = -p_t \mathbf{I} + \left(\lambda - \frac{2}{3}\mu \right) (\nabla \cdot \mathbf{v}) \mathbf{I} + 2\mu \mathbf{D} \tag{2.15}$$

p_t	Thermodynamic pressure	N/m^2
λ	Bulk viscosity	$kg/(ms)$
μ	Dynamic viscosity	$kg/(ms)$
$\mathbf{D}$	Rate of deformation tensor	s^{-1}

Table 2.3: Physical quantities of stress tensor.

The definition can be simplified by recalling the definition of the thermodynamic pressure [7]:

$$p_t = p - \left(\lambda + \frac{2}{3}\mu \right) (\nabla \cdot \mathbf{v}) \tag{2.16}$$

Inserting (2.16) into (2.15) leads to:

$$\sigma = -p\mathbf{I} + 2\lambda(\nabla \cdot \mathbf{v})\mathbf{I} + 2\mu \mathbf{D} \tag{2.17}$$

The first term deals with the bulk viscosity of the fluid and is related to the compression work. It may be neglected for the following reason. The influence of the bulk viscosity is bound for two extreme cases. On the one side, the volume viscosity vanishes for ideal gases $\lambda = 0$. On the other side, the mass conservation of incompressible fluids states $\nabla \cdot \mathbf{v} = 0$. By this fact it is possible to ignore its influence for polymers, whose properties lay between these two cases. The second term is the product of the dynamic viscosity of the fluid and the deformation tensor. The

rate of deformation tensor $\mathbf{D}$ is defined as the sum over the velocity gradients but will remain as $\mathbf{D}$ for simplicity.

$$\mathbf{D} = \frac{1}{2}\left((\nabla \mathbf{v}) + (\nabla \mathbf{v})^T\right) \tag{2.18}$$

Plugging the simplified stress tensor σ into the momentum equation yields the final form of the momentum equation:

$$\boxed{\rho \frac{\partial \mathbf{v}}{\partial t} + \rho \mathbf{v} \cdot \nabla \mathbf{v} = \rho \mathbf{g} - \nabla p + 2\nabla \cdot \mu \mathbf{D}} \tag{2.19}$$

In contrast to the presented balance equations, the energy equation is rather long in its derivation. It bases on the basic rule of energy conservation, which states that the total change of energy inside a closed system is equivalent to the sum of the work $\dot{w}$ and heating streams $\dot{q}$ applied to it.

$$\underbrace{\frac{1}{2}\frac{\partial}{\partial t}(\rho \mathbf{v} \cdot \mathbf{v}) + \frac{1}{2}\nabla \cdot (\rho \mathbf{v}\mathbf{v} \cdot \mathbf{v})}_{\text{Kinetic energy}} + \underbrace{\frac{\partial}{\partial t}(\rho \hat{U}) + \nabla \cdot (\rho \hat{U} \mathbf{v})}_{\text{Internal energy}} = \dot{w} + \dot{q} \tag{2.20}$$

The work is performed by the movement of the continuum inside a conservative potential field, like the gravitational field as well as by traction forces, including compression and shearing amongst others.

$$\dot{w} = \rho \mathbf{g} \cdot \mathbf{v} + \nabla \cdot (\sigma \cdot \mathbf{v}) \tag{2.21}$$

The heating can be reduced to two factors: temperature diffusion and the reactions heat Q_E:

$$\dot{q} = -\nabla \cdot \mathbf{q} + Q_E \tag{2.22}$$

Given the case, that the temperature linked effects on polymers are not considered on a microscopic scale, the Fourier's law of heat conduction can be applied as a valid assumption for the heat flux $\mathbf{q}$ [8]:

$$\mathbf{q} = -\kappa \nabla T \tag{2.23}$$

The scalar quantity κ stands for the thermal conductivity $[W/(mK)]$ and can be obtained from empirical data [4]. The second term, the reaction's heat Q_E, is a heat source present in the foam phase. It directly depends on the local reactants, water and polyol. The produced heat depends in the first place on the respective reaction enthalpies ΔH_{OH} and ΔH_W. These values resemble the heat quantities of each of the reactions that are liberated in a normed state. The enthalpies are then scaled based on the concentration of each of the chemicals and the conversion rates

Q_{OH} and Q_W that is some sense provides information about the reaction speed.

$$Q_E = \alpha \frac{\rho_F}{\rho_{F,0}} \left[\Delta H_{OH} c_{OH,0} Q_{OH} + \Delta H_W c_{W,0} Q_W \right] \tag{2.24}$$

The newly introduced scalar α represents the polymer fraction and has to be respected in this source term since only the polymer gives place to the reaction and not the surrounding air. The interphase model will be dealt with in the section 2.4. So far the thermal sources have been dealt with.

In order to further simplify the energy balance equation (2.20) one can employ the following two identities:

$$\begin{aligned} \frac{1}{2} \frac{\partial (\mathbf{v} \cdot \mathbf{v})}{\partial t} &= \mathbf{v} \cdot \frac{\partial \mathbf{v}}{\partial t} \\ \nabla \cdot (\sigma \cdot \mathbf{v}) - (\nabla \cdot \sigma) \cdot \mathbf{v} &= \sigma : (\nabla \mathbf{v}). \end{aligned} \tag{2.25}$$

Introducing both equations (2.25) into the energy conservation (2.20) together with the evaluated source terms yields a lengthy expression. Using the conservation of momentum some of the terms can be cancelled out.

$$\mathbf{v} \cdot \underbrace{\left(\frac{\partial}{\partial t}(\rho \mathbf{v}) + \nabla \cdot (\rho \mathbf{v}\mathbf{v}) \right)}_{\overset{(2.13)}{=} \rho \mathbf{g} + \nabla \cdot \sigma} + \frac{\partial}{\partial t}(\rho \hat{U}) + \nabla \cdot (\rho \hat{U} \mathbf{v}) = \rho \mathbf{g} \cdot \mathbf{v} + \nabla \cdot (\sigma \cdot \mathbf{v}) + \dot{q} \tag{2.26}$$

$$\frac{\partial}{\partial t}(\rho \hat{U}) + \nabla \cdot (\rho \hat{U} \mathbf{v}) = \nabla \cdot (\sigma \cdot \mathbf{v}) - (\nabla \cdot \sigma) \cdot \mathbf{v} + \nabla \cdot (\kappa \nabla T) + Q_E \tag{2.27}$$

$$\rho \frac{d\hat{U}}{dt} = \sigma : (\nabla \mathbf{v}) + \nabla \cdot (\kappa \nabla T) + Q_E \tag{2.28}$$

At this point, a valid representation of the internal energy $\hat{U}$ has to be recalled, as by itself it cannot be measured. It is nonetheless dependent on temperature and pressure, which are included in the other two driving equations (2.11) and (2.19). To proceed with the derivation, the differentials of enthalpy and entropy are recalled.

$$d\hat{H} = \left(\frac{\partial \hat{H}}{\partial p} \right)_T dp + \left(\frac{\partial \hat{H}}{\partial T} \right)_p dT \tag{2.29}$$

$$d\hat{S} = \left(\frac{\partial \hat{S}}{\partial p} \right)_T dp + \left(\frac{\partial \hat{S}}{\partial T} \right)_p dT \tag{2.30}$$

The relationship between entropy, enthalpy and energy reversible processes [9] are the following:

$$d\hat{U} = \delta\hat{Q} - \delta\hat{W} \tag{2.31}$$

$$\hat{H} = \hat{U} + p\hat{V} \tag{2.32}$$

$$\delta\hat{Q} = Td\hat{S} \tag{2.33}$$

$$\delta\hat{W} = pd\hat{V}. \tag{2.34}$$

Combining them leads to:

$$d\hat{S} = \frac{1}{T}\left[\left(\frac{\partial\hat{H}}{\partial p}\right)_T - \hat{V}\right]dp + \frac{1}{T}\left(\frac{\partial\hat{H}}{\partial T}\right)_p dT \tag{2.35}$$

Comparing (2.30) and (2.35) shows the following equivalence:

$$\left(\frac{\partial\hat{S}}{\partial p}\right)_T = \frac{1}{T}\left[\left(\frac{\partial\hat{H}}{\partial p}\right)_T - \hat{V}\right] \tag{2.36}$$

$$\left(\frac{\partial\hat{S}}{\partial T}\right)_p = \frac{1}{T}\left(\frac{\partial\hat{H}}{\partial T}\right)_p \tag{2.37}$$

To equate (2.36) and (2.37), each equation has to be respectively differentiated this respect to the temperature and pressure, leading to:

$$\left[\frac{\partial}{\partial T}\left(\frac{\partial\hat{S}}{\partial p}\right)_T\right]_p = \frac{1}{T^2}\left[-\left(\frac{\partial\hat{H}}{\partial p}\right)_T + \hat{V}\right] + \frac{1}{T}\left[\frac{\partial^2\hat{H}}{\partial T\partial p} - \left(\frac{\partial\hat{V}}{\partial T}\right)_p\right] = \frac{\partial^2\hat{S}}{\partial T\partial p} \tag{2.38}$$

$$\left[\frac{\partial}{\partial p}\left(\frac{\partial\hat{S}}{\partial T}\right)_p\right]_T = \frac{1}{T}\left(\frac{\partial^2\hat{H}}{\partial p\partial T}\right) = \frac{\partial^2\hat{S}}{\partial p\partial T} \tag{2.39}$$

Now joining (2.38) and (2.39) by the Schwarz's rule of symmetry we get the relation:

$$\left(\frac{\partial\hat{H}}{\partial p}\right)_T = \hat{V} + T\left(\frac{\partial\hat{V}}{\partial T}\right)_p \tag{2.40}$$

To shorten the formula the volume expansion coefficient is introduced. It is a factor which describes the isobaric expansion of a material on varying temperature.

$$\beta = \frac{1}{\hat{V}} \left(\frac{\partial \hat{V}}{\partial T} \right)_p \tag{2.41}$$

The temperature dependent coefficient β is calculated from measurement data with the help of numerical interpolation. Additionally (2.40) is plugged into (2.29).

$$d\hat{H} = (1 - \beta T)\hat{V} dp + \left(\frac{\partial \hat{H}}{\partial T} \right)_p dT \tag{2.42}$$

The term $\left(\frac{\partial \hat{H}}{\partial T}\right)_p$ corresponds to the definition of the specific isobaric heat capacity c_p and hence will be substituted. The heat capacity too is strongly dependent on the stage of the foaming and needs to be extracted from measurement tables.

$$d\hat{H} = (1 - \beta T)\hat{V} dp + c_p dT \tag{2.43}$$

By (2.32) the enthalpy can be rewritten as:

$$d(\hat{U} + p\hat{V}) = d\hat{U} + p d\hat{V} + \hat{V} dp = (1 - \beta T)\hat{V} dp + c_p dT \tag{2.44}$$

$$d\hat{U} = -\beta T \hat{V} dp + c_p dT - p d\hat{V} \tag{2.45}$$

The differentials are now set specifically as time derivatives:

$$\frac{d\hat{U}}{dt} = -\beta T \hat{V} \frac{dp}{dt} + c_p \frac{dT}{dt} - p \frac{d\hat{V}}{dt} \tag{2.46}$$

Since the density ρ is the preferred quantity for the mass volume relation, the specific volume $\hat{V}$ will be substituted by $\hat{V} = \rho^{-1}$.

$$\frac{d\hat{V}}{dt} = \frac{\partial \hat{V}}{\partial \rho} \frac{d\rho}{dt} = -\frac{1}{\rho^2} \left(\frac{\partial \rho}{\partial t} + \mathbf{v} \cdot \nabla \rho \right) \tag{2.47}$$

Now a zero addition will be done with the goal of eliminating the time derivative of the density.

$$-\frac{1}{\rho^2} \underbrace{\left(\frac{\partial \rho}{\partial t} + \mathbf{v} \cdot \nabla \rho + \rho (\nabla \cdot \mathbf{v}) \right)}_{\overset{(2.11)}{=} 0} + \frac{1}{\rho} (\nabla \cdot \mathbf{v}) = \frac{1}{\rho} (\nabla \cdot \mathbf{v}) \tag{2.48}$$

$$\rho \frac{d\hat{U}}{dt} = -\beta T \frac{dp}{dt} + \rho c_p \frac{dT}{dt} - p(\nabla \cdot \mathbf{v}) \tag{2.49}$$

The total difference of the internal energy (2.49) can now be inserted into (2.28).

$$\rho c_p \frac{dT}{dt} = \beta T \frac{dp}{dt} + \sigma : (\nabla \mathbf{v}) + p(\nabla \cdot \mathbf{v}) + \nabla \cdot (\kappa \nabla T) + Q_E \tag{2.50}$$

Finally some simplifications are performed on the stress related term:

$$\begin{aligned} \sigma : \nabla \mathbf{v} &\overset{(2.15)}{=} (-p\mathbf{I} + 2\mu \mathbf{D}) : \nabla \mathbf{v} = -p(\nabla \cdot \mathbf{v}) + 2\mu \left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T\right) : \nabla \mathbf{v} && (2.51) \\ &= -p(\nabla \cdot \mathbf{v}) + \mu \left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T\right) : \left(\nabla \mathbf{v} + (\nabla \mathbf{v})^T\right) && (2.52) \end{aligned}$$

The friction related expression $\sqrt{(\nabla \mathbf{v} + (\nabla \mathbf{v})^T) : (\nabla \mathbf{v} + (\nabla \mathbf{v})^T)}$ is often denoted as the shear rate $\dot{\gamma}\,[s^{-1}]$, a quantity describing the drift between neighbouring fluid particles.

$$\underbrace{\rho c_p \frac{dT}{dt}}_{I} = \underbrace{\beta T \frac{dp}{dt}}_{II} + \underbrace{\mu \dot{\gamma}^2}_{III} + \underbrace{\nabla \cdot (\kappa \nabla T)}_{IV} + \underbrace{Q_E}_{V} \tag{2.53}$$

The current shape of the energy equation shows, that the total change in (volume-specific) thermal energy, described by I, equals the compression power II plus the shear heating power and the thermal diffusion power. It is often the case that the shear heating (III) is neglected due to its comparatively small effect. By expanding the total derivatives the final form of the energy equation is presented:

$$\rho c_p \left(\frac{\partial T}{\partial t} + \mathbf{v} \cdot \nabla T\right) = \beta T \left(\frac{\partial p}{\partial t} + \mathbf{v} \cdot \nabla p\right) + \mu \dot{\gamma}^2 + \nabla \cdot (\kappa \nabla T) + Q_E \tag{2.54}$$

2.4 Phase Modelling

As stated before, the process will be simplified to a two-phase flow featuring the foam on one side and air on the other one. The interface will be modelled using the Volume-of-Fluid (VOF) method, which assumes the liquids to be immiscible and their interface to be a sharp surface. The scalar field $\alpha \in [0, 1]$ is used to delimit the regions filled with foam ($\alpha = 1$) or air ($\alpha = 1$). Values between 0 and 1 denote the interface zone of both liquids. The parameter function $\alpha(t, \mathbf{x})$ can then be modelled using the conservation equation:

$$\frac{\partial \alpha}{\partial t} + \nabla \cdot \alpha \mathbf{v} = 0 \tag{2.55}$$

A further advantage of using the alpha phase is the simple way in which material properties are set. The material-dependent values as for instance the viscosity μ or density ρ can be expressed as a convex combination of the ones of air and polymer depending on the locally present α values. In the case of the dynamic viscosity for instance (A=Air, F=Foam):

$$\mu = \alpha \mu_F + (1 - \alpha)\mu_A \tag{2.56}$$

Each phase also has its own material laws. For most applications assuming the air phase to be an ideal gas suffices since the pressure exerted on the air rarely exceeds 5 bar and its temperature is always above room temperature ($293K$). Under these assumptions, the modelling error of the air phase is regarded as acceptable [10]. Further facts related to the material model of ideal gases are well documented and explained in the initial sections of [10] and will be omitted for the sake of brevity.

The polymer phase requires a few additional transport equations related to the conversion fields X_{OH} and X_W. As seen in the section of the reaction dynamics, these quantities are locally relevant for the density transition as well as the heat production. As parts of the foam, they are transported through the domain and are subject to conservation laws. To derive these, it suffices to insert $\rho\alpha X$ as the transported entity F in (2.10). Due to the chemical reaction, it is necessary to have a sink term on the right-hand side G:

$$\frac{\partial \alpha \rho X_i}{\partial t} + \nabla \cdot (\alpha \rho X_i \mathbf{v}) = \rho \alpha \frac{dX_i}{dt} \tag{2.57}$$

Here again it is possible to simplify the equation to its convective form using the mass continuity (2.11):

$$\rho \left(\frac{\partial \alpha X_i}{\partial t} + \mathbf{v} \cdot \nabla(\alpha X_i) \right) + \alpha X_i \underbrace{\left(\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \mathbf{v}) \right)}_{\overset{(2.11)}{=} 0} = \rho \alpha Q_i \tag{2.58}$$

$$\frac{\partial \alpha X_i}{\partial t} + \mathbf{v} \cdot \nabla(\alpha X_i) = \alpha Q_i \tag{2.59}$$

Plugging in X_{OH} and X_W yields the final form of the equations:

$$\frac{\partial \alpha X_{OH}}{\partial t} + \mathbf{v} \cdot \nabla(\alpha X_{OH}) = \alpha Q_{OH} \tag{2.60}$$

$$\frac{\partial \alpha X_W}{\partial t} + \mathbf{v} \cdot \nabla(\alpha X_W) = \alpha Q_W \tag{2.61}$$

2.5 Boundary Conditions

The boundary conditions of foaming procedures are complex to model if all the details have to be taken into account. In practice, the injection nozzle travels through an opened section of the mould until the injection is completed. Shortly after, the mould is covered and sealed by the missing plate. An accurate representation of this action requires dynamic meshes that cause a significant increase in computation cost. This level of detail is not required for the given optimization. Instead, boundaries with varying conditions will be used that mimic the filling procedure. The types of boundary found in a mould are the following: inlet, outlet or wall. For each of these boundaries it holds that if there is a prescribed velocity, the pressure will be set as a homogeneous Neumann condition. In the same way, the velocity is defined by $\mathbf{n} \cdot \nabla \mathbf{v} = \mathbf{0}$ (hom. Neumann) whenever a surface has a fixed pressure condition. The alpha phase is set to 1 on the injection location and to $\mathbf{n} \cdot \nabla \alpha = 0$ elsewhere.

1. Inlet: The inlet faces of the mould cover the whole surface that can contain a valid injection location. The user can define such a location on which the round cross-section of the injection nozzle is cast onto. On this circular patch, a parabolic velocity profile is set as a Dirichlet condition.

2. Outlet: Any of the faces that are meant to let the air out, but still retain polymer is set to be an outlet including the inlet's surface that is not covered by the nozzle. The outlet works as a sealing boundary that remains open as long as no polymer gets into its proximity. The α phase, therefore, determines the behaviour of the boundary. The air is let out of the mould driven by a fixed environment pressure when the surrounding α field is 0. Once the polymer has reached a certain distance threshold, the cells are sealed enforcing no-slip conditions on the velocity.

3. Wall: All of the remaining surfaces are walls. They are permanently sealed with no-slip conditions and a zero pressure gradient.

A small demonstration of the working boundary conditions can be seen in figure 2.3. A subset of one geometry is shown during the polymer injection. The front- and back faces have been set opaque to better see the polymer flow in the inside. The magnitude of the velocity vector is shown with the help of the colour scale. In the center top position of the figure one can see the parabolic velocity profile of the injection nozzle. In an area around it, the air is blocked from exiting the mould due to the polymer's proximity. Further away there is a perceivable velocity at the boundaries, which is attributed to the escaping air. The translucid faces are walls while the other surfaces are partly inlet and outlet faces depending on whether they are theoretically suited for injection.

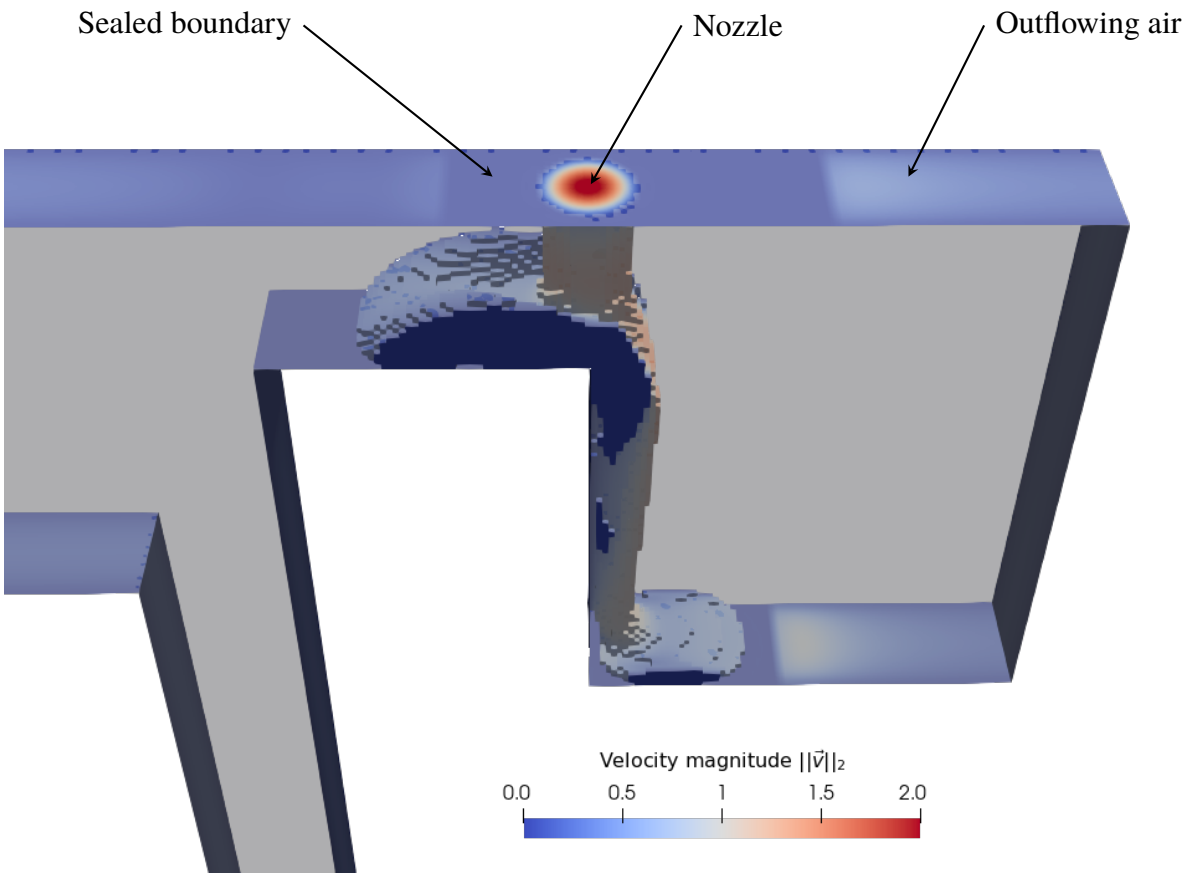

Figure 2.3: Injection recorded at $t = 0.4s$.

Chapter 3

Artificial Neural Networks

Artificial neural networks provide the foundation for the prediction model. The topic will be introduced starting with the inspiration found in biological neurons and will progressively converge towards nowadays's complex deep neural networks.

3.1 The Perceptron

The foundation for the nowadays popular artificial neural networks dates back to the 1950s, where mathematicians tried to mimic the functionality of a single neuron with a computing unit. From a simplified perspective, a biological neuron is a cell equipped with a long cable-like structure, called the axon and a set of dendrites. The filament-like dendrites are located around the core of the cell and are responsible for transmitting electro-chemical signals from surrounding nerve cells to the neuron. There, the soma gathers all the incoming impulses and releases a signal as soon as an activation threshold is reached. The axon then conducts the produced output signal away from the soma towards other cells. The junction between the axon terminal and the dendrites of other cells is called a synapse and it is the key to the formation of memory. The strength of its connection defines how much influence an input has on a neuron. This way inputs from our sensory organs are processed and passed along countless cells until they result in an action or response to it.

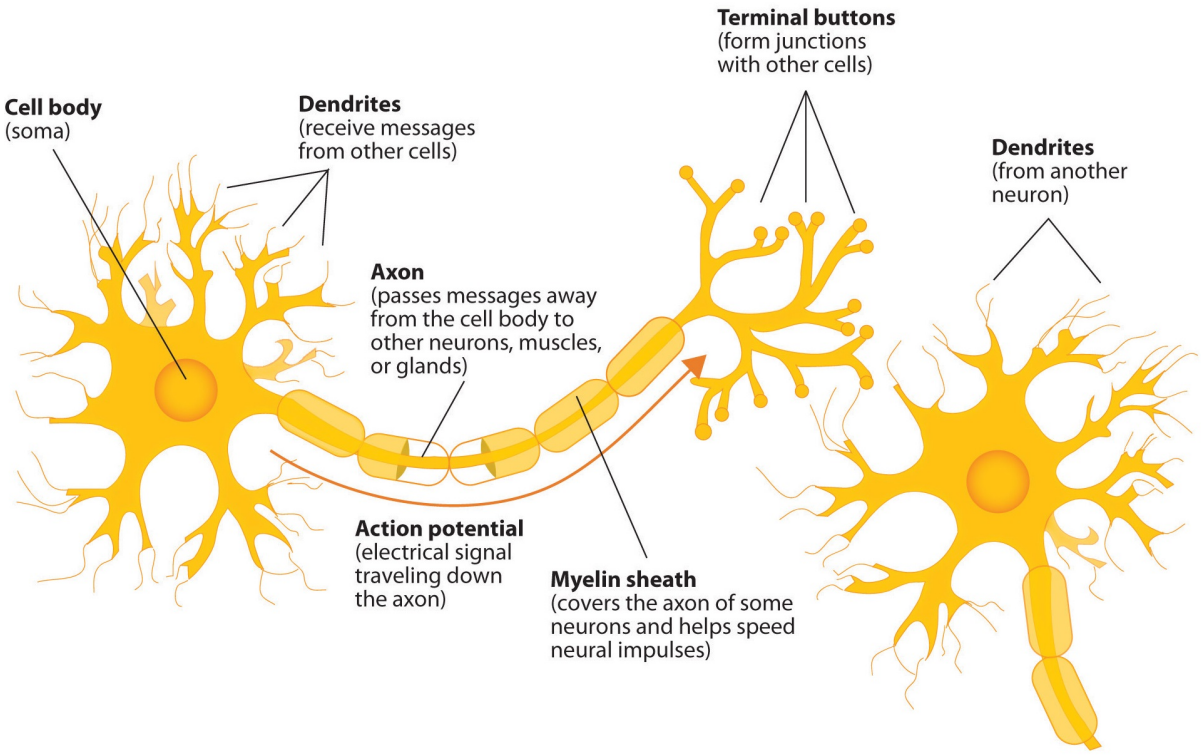

Figure 3.1: Conceptual view of neurons[11].

The first mathematical imitation of a neuron was called the Perceptron. Similar to their biological counterpart, the perceptron is set up to gather a set of inputs $\mathbf{x}$ that once multiplied by weights $\mathbf{w}$ are added up and passed through an activation function g. In addition to these, perceptrons also use a trainable bias which in some sense resembles the base potential of a neuron. This abstraction from biology is an oversimplification of the thoroughly complex and intricate processes of a brain, but nevertheless, it is regarded as the artificial neuron. In summary, the arithmetic of a perceptron can be expressed by:

$$g(\mathbf{w} \cdot \mathbf{x} + \theta) = y, \qquad \mathbf{x}, \mathbf{w} \in \mathbb{R}^n \tag{3.1}$$

For simplicity, the bias is merged into the feature vector as follows:

$$\mathbf{w} = (\theta, w_1, \ldots, w_n) \qquad \mathbf{x} = (1, x_1, \ldots, x_n)$$

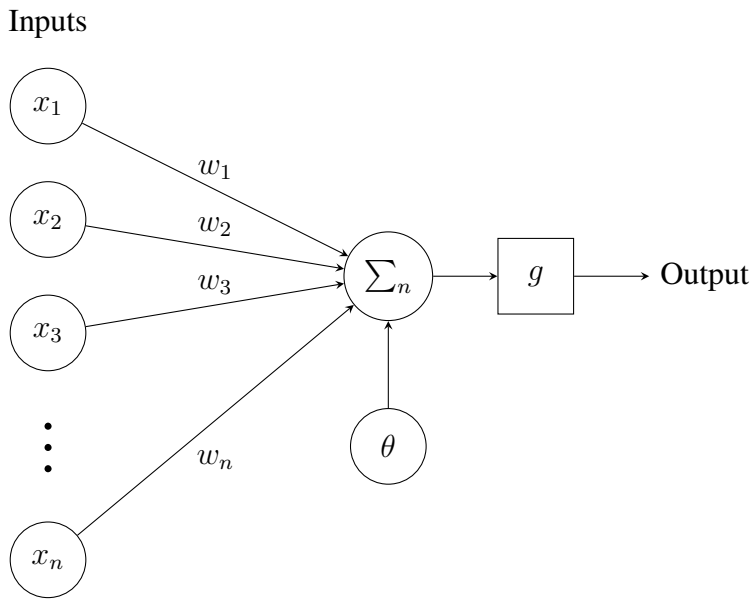

Figure 3.2: Single perceptron.

A perceptron can be set up to solve different problems of the field of supervised learning such as binary classification and linear regression. The functionality depends primarily on the choice of error metric and the activation function g. Some simple examples for activation functions and their applications are:

1. Binary classification: $g(x) := \text{sign}(x)$
2. Linear regression: $g(x) = x$

While the activation function is part of the Perceptron, the error metric is rather a measure to help train the weights. For a well-posed problem, a dataset with training samples is required. The data samples contained in it have an array of features, i.e. quantities describing the sample, and labels that ought to be inferred by the predictor. The mathematical notation for the training set is given by $\mathcal{D} = \{(\mathbf{x}_k, y_k) | k = 1, \ldots, n\}$. The error function $L(\mathbf{w})$ then quantifies how far the predictions are from the correct labels. As stated before, a certain error function may suit one sort of problem more than another one. In face of an abundance of different metrics a proper choice may not be trivial. A possible match-up for the classification and regression problems stated before can be:

1. Binary classification (Binary Cross-Entropy):

$$L(\mathbf{w}) = -\frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} y_d \log(g(\mathbf{w}, \mathbf{x}_d)) + (1 - y_d) \log(1 - g(\mathbf{w}, \mathbf{x}_d)))$$

2. Linear regression (Mean Squared Error):

$$L(\mathbf{w}) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} (y_d - g(\mathbf{w}, \mathbf{x}_d))^2$$

To avoid an artificially high error when having large sample sets, the error functions are averaged over the dataset size $|\mathcal{D}|$. The training problem is then defined as finding the set of weights $\mathbf{w}$ such that the error is minimized:

$$\mathbf{w}^* = \arg\min_{\mathbf{w} \in \mathbb{R}^{n+1}} L(\mathbf{w}) \tag{3.2}$$

An open question now remains how to obtain the proper weight factors $\mathbf{w}$. For the case of the perceptron, several possible algorithms can be employed. The most basic one is the perceptron learning algorithm, suited for linear classification, i.e. classification with linear separating hyperplanes. While simple and of linear complexity, it is only sure to converge for linearly separable sets. In mathematical terms, this means that for all the labels in a set of samples $y_i \in \{1, \ldots, k\}$ the convex hull spanned by their samples $\mathbf{x}_i$ do not intersect with the other classes $(\mathbf{x}_j, y_j)$:

$$C(\mathbf{x}_i) \cap C(\mathbf{x}_j) = \emptyset \qquad \forall\, i, j \in \{1, \ldots, k\}, i \neq j \tag{3.3}$$

Linear separability is rarely given in common problems. Apart from most problems being more complex, both noise and outliers in the data samples can prevent the perceptron algorithm from converging. A good remedy for these issues is applying a more versatile, robust algorithm, the Delta rule. Instead of aiming to fit the data perfectly, this algorithm tries to minimize the error function L with gradient descend steps. Define η as a small step size (typically around 0.01) and denote with $\nabla_{\mathbf{w}}$ the gradient with respect to the weights. Then the update rule for small randomly initiated weights $\mathbf{w}^0$ is:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} L(\mathbf{w}^t) \tag{3.4}$$

For the sake of demonstration, consider an example case with the sigmoid function $\sigma(x)$:

$$g(x) = \frac{1}{1 + e^{-x}} = \sigma(x) \tag{3.5}$$

$$L(\mathbf{w}) = \frac{1}{2} \sum_{d \in \mathcal{D}} (y_d - g(\mathbf{w} \cdot \mathbf{x}_d))^2 \tag{3.6}$$

To compute the learning rule, first calculate the term $\nabla_{\mathbf{w}} L$ and determine the required components.

$$\begin{aligned}\nabla_{\mathbf{w}} L &= \frac{1}{2}\sum_{d\in\mathcal{D}} \nabla_{\mathbf{w}}(y_d - g(\mathbf{w}\cdot\mathbf{x}_d))^2 = \sum_{d\in\mathcal{D}}(y_d - g(\mathbf{w}\cdot\mathbf{x}_d))\nabla_{\mathbf{w}}(y_d - g(\mathbf{w}\cdot\mathbf{x}_d)) \\ &= -\sum_{d\in\mathcal{D}}(y_d - g(\mathbf{w}\cdot\mathbf{x}_d))\mathbf{x}_d g'(\mathbf{w}\cdot\mathbf{x}_d)\end{aligned} \tag{3.7}$$

After computing the derivative of g, all needed identities are known for the algorithm.

$$g'(x) = \frac{e^x}{(1+e^x)^2} = \sigma(x)(1-\sigma(x)) \tag{3.8}$$

The Gradient Descend method (GD), on which the Delta rule bases, is a key actor in solving the optimization problem. It provides the foundation of the learning procedures of both simple Perceptrons and extensive ANN and thus it will be analyzed together with its popular variations found in Machine Learning.

3.2 Gradient Descend

As mentioned before, many popular ML algorithms boil down to solving a minimization problem of the shape (3.9). Hereby, it is assumed that there exists a simply connected subset $\mathcal{B} \cup \mathbb{R}^n$ that contains at least one local minimum $\mathbf{w}^*$.

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathcal{B}} f(\mathbf{w}) \tag{3.9}$$

This is not just the case of the Perceptron, but also other seemingly unrelated algorithms like the Support Vector Machine and the Least Squares method. Hence, it is of considerable interest to find robust and low complexity algorithms to solve them.

The GD method is one of the oldest and most used ones if its variations are considered. It derived from the thought that in order to find a minimum over a function f one has to progressively walk down the path of the steepest descend to approach the minimum at $\mathbf{w}^*$. The sequence of iterates of GD is given by:

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta\nabla f(\mathbf{w}^t) \tag{3.10}$$

for a small enough positive step size η. The convergence of the GD method depends directly on the assumptions made on the function f. Consequently, the proofs yield different convergence speeds for each case. Before proceeding with the convergence properties and proofs, a short overview of the notation and definitions used:

$\mathbf{w}$	Weight vector
$\mathbf{w}^*$	Optimal weight vector
f	Function, e.g. loss function
λ	Regularization factor
L	Lipschitz bound
ξ	Strong convexity bound
η	Step size in GD methods
t	Iteration index
$\|.\|_p$	p-norm

Table 3.1: Notation used in supervised learning.

One of the key assumptions used is the Lipschitz smoothness condition. In a one-dimensional example, this condition can be visualized with a cone spanned by the functions g_1 and g_2.

$$\begin{aligned} g_1(x) &= L(x - \tilde{x}) + f(\tilde{x}) \\ g_2(x) &= -L(x - \tilde{x}) + f(\tilde{x}) \end{aligned} \tag{3.11}$$

For an L-smooth function, there has to exist an L such that the cone only intersects the derivative at $x = \tilde{x}$. This is in some sense a bound on the gradient with respect to step size in x. It can be better visualized in figure 3.3, where a generic derivative ∇f is sketched together with the cone spanned by the functions g_1 and g_2. With this visualization it is easy to see that any continuous derivative that can be bound by a linear function is indeed L-smooth.

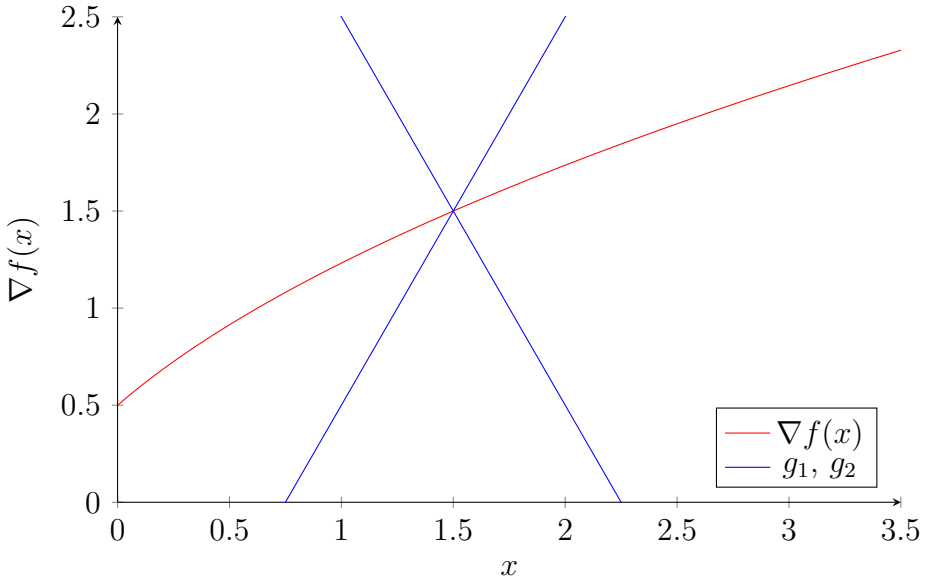

Figure 3.3: Visualization of Lipschitz smoothness.

Definition 1 (L-smoothness)**.** A function $f : \mathcal{B} \to \mathbb{R}$ is L-smooth whenever there exist an $L \in \mathbb{R}_{\geq 0}$ such that for any $\mathbf{w}, \mathbf{y} \in \mathcal{B}$:

$$L \|\mathbf{w} - \mathbf{y}\| \geq \|\nabla f(\mathbf{w}) - \nabla f(\mathbf{y})\| \tag{3.12}$$

Assuming that the f is also twice differentiable, it is possible to formulate a bound that resembles in some sense a Taylor expansion of degree two. This relationship will be of substantial use when proving the convergence of the gradient descend methods that will be introduced below.

Lemma 1 (Implication of Lipschitz smoothness)**.** *A L-smooth function f that is twice differentiable in $\mathcal{B}$ satisfies:*

$$\mathbf{y}^T(\nabla^2 f(\mathbf{w}))\mathbf{y} \leq L \|\mathbf{y}\|_2^2 \tag{3.13}$$

and as a consequence also

$$f(\mathbf{y}) \leq f(\mathbf{w}) + (\mathbf{y} - \mathbf{w})^T \nabla f(\mathbf{w}) + \frac{L}{2} \|\mathbf{y} - \mathbf{w}\|_2^2 \tag{3.14}$$

for all $\mathbf{w}, \mathbf{y} \in \mathcal{B}$.

Proof.

Using the expansion theorem 1, develop a first order approximation of the gradient of f around $\mathbf{w} \in \mathbb{R}^n$. $\mathbf{y}$ is introduced as $\mathbf{y} = \mathbf{w} + \eta\mathbf{v}$.

$$\nabla f(\mathbf{w}) = \nabla f(\mathbf{w} + \eta\mathbf{v}) + \int_{\tau=0}^{\eta} \nabla^2 f(\mathbf{w} + \tau\mathbf{v})\mathbf{v}\, d\tau \tag{3.15}$$

Next take the dot product with $\mathbf{v}$ and apply the triangle inequality:

$$\mathbf{v}^T \left(\nabla f(\mathbf{w}) - \nabla f(\mathbf{w} + \eta\mathbf{v})\right) = \int_{\tau=0}^{\eta} \mathbf{v}^T \nabla^2 f(\mathbf{w} + \tau\mathbf{v})\mathbf{v}\, d\tau \tag{3.16}$$

$$\int_{\tau=0}^{\eta} \mathbf{v}^T \nabla^2 f(\mathbf{w} + \tau\mathbf{v})\mathbf{v}\, d\tau \leq \|\mathbf{v}\|_2 \|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w} + \eta\mathbf{v})\|_2 \tag{3.17}$$

Using the Lipschitz property 3.12, the right hand side simplifies to:

$$\|\mathbf{v}\|_2 \|\nabla f(\mathbf{w}) - \nabla f(\mathbf{w} + \eta\mathbf{v})\|_2 \leq \|\mathbf{v}\|_2 L \|\eta\mathbf{v}\|_2 = \eta L \|\mathbf{v}\|_2^2 \tag{3.18}$$

Finally, both sides are divided by η and the limit $\eta \to 0$ is taken:

$$\lim_{\eta \to 0} \int_{\tau=0}^{\eta} \mathbf{v}^T \nabla^2 f(\mathbf{w} + \tau \mathbf{v}) \mathbf{v} \, d\tau \leq L \left\| \mathbf{v} \right\|_2^2 \tag{3.19}$$

By the Mean Value Theorem, the left term simplifies to:

$$\mathbf{v}^T \nabla^2 f(\mathbf{w}) \mathbf{v} \leq L \left\| \mathbf{v} \right\|_2^2 \tag{3.20}$$

The second part of the lemma is proven by again using the first order expansion and performing a 0 addition. Take $\mathbf{w}, \mathbf{y} \in \mathbb{R}^n$:

$$f(\mathbf{w}) = f(\mathbf{y}) + (\mathbf{w} - \mathbf{y})^T \nabla f(\mathbf{y}) + \int_{\tau=0}^{1} (\mathbf{w} - \mathbf{y})^T (\nabla f(\mathbf{y} + \tau(\mathbf{w} - \mathbf{y})) - \nabla f(\mathbf{y})) \, d\tau \tag{3.21}$$

Now use the fact that

$$\int_{\tau=\tau_0}^{\tau_1} g(\tau) \, d\tau \leq \int_{\tau=\tau_0}^{\tau_1} \left| g(\tau) \right| \, d\tau \tag{3.22}$$

on equation (3.21).

$$f(\mathbf{w}) \leq f(\mathbf{y}) + (\mathbf{w} - \mathbf{y})^T \nabla f(\mathbf{y}) + \int_{\tau=0}^{1} \left| (\mathbf{w} - \mathbf{y})^T (\nabla f(\mathbf{y} + \tau(\mathbf{w} - \mathbf{y})) - \nabla f(\mathbf{y})) \right| \, d\tau \tag{3.23}$$

Now the triangle inequality is again applied to the terms in the integral together:

$$f(\mathbf{w}) \leq f(\mathbf{y}) + (\mathbf{w} - \mathbf{y})^T \nabla f(\mathbf{y}) + \left\| \mathbf{w} - \mathbf{y} \right\|_2 \int_{\tau=0}^{1} \left\| \nabla f(\mathbf{y} + \tau(\mathbf{w} - \mathbf{y})) - \nabla f(\mathbf{y}) \right\|_2 \, d\tau \tag{3.24}$$

Finally, using the L-smoothness property (3.12) brings the equation to the desired statement.

$$f(\mathbf{w}) \leq f(\mathbf{y}) + (\mathbf{w} - \mathbf{y})^T \nabla f(\mathbf{y}) + L \left\| \mathbf{w} - \mathbf{y} \right\|_2^2 \int_{\tau=0}^{1} \tau \, d\tau \tag{3.25}$$

$$f(\mathbf{w}) \leq f(\mathbf{y}) + (\mathbf{w} - \mathbf{y})^T \nabla f(\mathbf{y}) + \frac{L}{2} \left\| \mathbf{w} - \mathbf{y} \right\|_2^2 \tag{3.26}$$

q.e.d

The second important definition that will be used is the concept of convexity. Much like in the case of convex geometries or sets, convex functions are defined by the rule that a convex combination of evaluated points of the function has to be greater or equal to any value in a straight

line between them. An example for a convex function for instance is $f(x) = x^2$.

Definition 2 (Convexity)**.** The function $f : \mathbb{R}^n \to \mathbb{R}$ is called convex if for $\forall \mathbf{w}, \mathbf{y} \in \mathbb{R}^n$ and $t \in [0, 1]$ it holds that:

$$f(t\mathbf{w} + (1-t)\mathbf{y}) \leq tf(\mathbf{w}) + (1-t)f(\mathbf{y}) \tag{3.27}$$

If f is twice continuously differentiable $f \in C^2(\mathbb{R}^n)$, then convexity implies that the Hessian matrix of f is positive semi-definite for all $\mathbf{y}$ in the convex set:

$$\mathbf{w}^T \left(\nabla^2 f(\mathbf{y})\right) \mathbf{w} \geq 0 \tag{3.28}$$

A short lemma following the definition of convexity will be used.

Lemma 2 (Implication of convexity)**.** *A convex function $f : \mathcal{B} \to \mathbb{R}$ also satisfies:*

$$f(\mathbf{y}) \geq f(\mathbf{w}) + (\mathbf{y} - \mathbf{w})^T \nabla f(\mathbf{w}), \qquad \forall \mathbf{w}, \mathbf{y} \in \mathbb{R}^n \tag{3.29}$$

Proof.
Take (3.27) and divide both sides by t.

$$\frac{f(t\mathbf{w} + (1-t)\mathbf{y})}{t} \leq f(\mathbf{w}) + \frac{1-t}{t} f(\mathbf{y}) \tag{3.30}$$

$$\frac{f(\mathbf{y} + t(\mathbf{w} - \mathbf{y})) - f(\mathbf{y})}{t} \leq f(\mathbf{w}) - f(\mathbf{y}) \tag{3.31}$$

Taking the limit $t \to 0$ results in the derivative in $\mathbf{w} - \mathbf{y}$ direction:

$$\lim_{t \to 0} \frac{f(\mathbf{y} + t(\mathbf{w} - \mathbf{y})) - f(\mathbf{y})}{t} = (\mathbf{w} - \mathbf{y}) \cdot \nabla f(\mathbf{y}) \tag{3.32}$$

q.e.d

Both presented lemmas can now be used to derive one of the most common formulas in convex smooth optimization, the so-called Co-coercivity Lemma [12].

Lemma 3 (Co-coercivity)**.** *For a convex and L-smooth function f and $\mathbf{w}, \mathbf{y} \in \mathcal{B}$ it holds:*

$$(\mathbf{y} - \mathbf{w})^T \left(\nabla f(\mathbf{y}) - \nabla f(\mathbf{w})\right) \geq \frac{1}{L} \left\|\nabla f(\mathbf{y}) - \nabla f(\mathbf{w})\right\|_2^2 \tag{3.33}$$

Proof.

Start with (3.33) and perform a zero addition with $\mathbf{z} = \mathbf{w} - \frac{\nabla f(\mathbf{w}) - \nabla f(\mathbf{y})}{L}$:

$$f(\mathbf{y}) - f(\mathbf{w}) = f(\mathbf{y}) - f(\mathbf{z}) + f(\mathbf{z}) - f(\mathbf{w}) \tag{3.34}$$

The second equation of lemma 3.12 (3.26) can be used as follows:

$$f(\mathbf{z}) - f(\mathbf{w}) \leq (\mathbf{z} - \mathbf{w})^T \nabla f(\mathbf{w}) + \frac{L}{2} \|\mathbf{z} - \mathbf{w}\|_2^2 \tag{3.35}$$

together with the convexity property (3.29):

$$f(\mathbf{y}) - f(\mathbf{z}) \leq (\mathbf{y} - \mathbf{z})^T \nabla f(\mathbf{y}) \tag{3.36}$$

Plugging both upper bounds into (3.34) yields:

$$f(\mathbf{y}) - f(\mathbf{w}) \leq (\mathbf{y} - \mathbf{z})^T \nabla f(\mathbf{y}) + (\mathbf{z} - \mathbf{w})^T \nabla f(\mathbf{w}) + \frac{L}{2} \|\mathbf{z} - \mathbf{w}\|_2^2 \tag{3.37}$$

Now $\mathbf{z}$ is set to $\mathbf{z} = \mathbf{w} - \frac{1}{L}((\nabla f(\mathbf{w}) - \nabla f(\mathbf{y}))$, with the goal of tightening the inequality in (3.37).

$$\begin{aligned} (\mathbf{y} - \mathbf{w})^T \nabla f(\mathbf{y}) - \frac{1}{L} \|\nabla f(\mathbf{w}) - \nabla f(\mathbf{y})\|_2^2 + \frac{1}{2L} \|\nabla f(\mathbf{w}) - \nabla f(\mathbf{y})\|_2^2 \\ = (\mathbf{y} - \mathbf{w})^T \nabla f(\mathbf{y}) - \frac{1}{2L} \|\nabla f(\mathbf{w}) - \nabla f(\mathbf{y})\|_2^2 \end{aligned} \tag{3.38}$$

$$f(\mathbf{y}) - f(\mathbf{w}) \leq (\mathbf{y} - \mathbf{w})^T \nabla f(\mathbf{y}) - \frac{1}{2L} \|\nabla f(\mathbf{w}) - \nabla f(\mathbf{y})\|_2^2 \tag{3.39}$$

The final result can be derived by taking (3.39) and evaluating it in two different arrangements of $\mathbf{w}$ and $\mathbf{y}$:

$$f(\mathbf{y}) - f(\mathbf{w}) \leq (\mathbf{y} - \mathbf{w})^T \nabla f(\mathbf{y}) - \frac{1}{2L} \|\nabla f(\mathbf{w}) - \nabla f(\mathbf{y})\|_2^2 \tag{3.40}$$

$$f(\mathbf{w}) - f(\mathbf{y}) \leq (\mathbf{w} - \mathbf{y})^T \nabla f(\mathbf{w}) - \frac{1}{2L} \|\nabla f(\mathbf{y}) - \nabla f(\mathbf{w})\|_2^2 \tag{3.41}$$

Adding both equations (3.40), (3.41) returns the desired expression:

$$0 \leq (\mathbf{w} - \mathbf{y})^T (\nabla f(\mathbf{w}) - \nabla f(\mathbf{y})) - \frac{1}{L} \|\nabla f(\mathbf{w}) - \nabla f(\mathbf{y})\|_2^2 \tag{3.42}$$

$$\frac{1}{L}\left\|\nabla f(\mathbf{w})-\nabla f(\mathbf{y})\right\|_2^2 \leq (\mathbf{w}-\mathbf{y})^T\left(\nabla f(\mathbf{w})-\nabla f(\mathbf{y})\right) \tag{3.43}$$

q.e.d

With the derived relationships, it is now possible to prove the convergence of GD swiftly.

Theorem 4 (Convergence of GD)**.** *Let $f:\mathcal{B}\to\mathbb{R}$ be a twice differentiable function that is both convex and L-smooth on $\mathcal{B}$. Further, let $\mathbf{w}\in\mathcal{B}$ be an initial guess and η a real valued small scalar satisfying $\eta\in(0,L^{-1}]$. Then the iteration* (3.10) *for $n=2,\ldots,t$ will converge towards the local minimum at w^* following:*

$$f(\mathbf{w}^t)-f(\mathbf{w}^*)\leq\frac{2L}{t-1}\left\|\mathbf{w}^1-\mathbf{w}^*\right\|_2^2 \tag{3.44}$$

Proof.

Begin by proving that the sequence of steps towards the minimum is a decreasing sequence in t:

$$\begin{aligned}\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2 &= \left\|\mathbf{w}^t-\eta\nabla f(\mathbf{w}^t)-\mathbf{w}^*\right\|_2^2\\ &= \left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2-2\eta(\mathbf{w}^t-\mathbf{w}^*)^T\nabla f(\mathbf{w}^t)+\eta^2\left\|\nabla f(\mathbf{w}^t)\right\|_2^2\end{aligned} \tag{3.45}$$

To simplify the expression, the co-coercivity property (3.33) may be applied on the intermediate term on the right-hand side of equation (3.45).

$$-(\mathbf{w}^t-\mathbf{w}^*)^T\left(\nabla f(\mathbf{w}^t)-\nabla f(\mathbf{w}^*)\right)\leq-\frac{1}{L}\left\|\nabla f(\mathbf{w}^t)\right\|_2^2 \tag{3.46}$$

$$\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2\leq\left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2+\left(\eta^2-\frac{2\eta}{L}\right)\left\|\nabla f(\mathbf{w}^t)\right\|_2^2 \tag{3.47}$$

It can be observed that for any $\eta\in(0,L^{-1}]$ the sequence of steps will be decreasing in t:

$$\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2\leq\left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2\leq\ldots\leq\left\|\mathbf{w}^1-\mathbf{w}^*\right\|_2 \tag{3.48}$$

This fact may be used later on to relate the convergence speed to the error at the initialization step. Next, relate the function value at a time t witht the value of the minimum using the convexity property (3.29):

$$\begin{aligned}f(\mathbf{w}^*)&\geq f(\mathbf{w}^t)+(\mathbf{w}^*-\mathbf{w}^t)^T\nabla f(\mathbf{w}^t)\\ f(\mathbf{w}^t)-f(\mathbf{w}^*)&\leq(\mathbf{w}^t-\mathbf{w}^*)^T\nabla f(\mathbf{w}^t)\end{aligned} \tag{3.49}$$

Applying the triangle inequality and equation (3.48) results in:

$$\begin{aligned} f(\mathbf{w}^t) - f(\mathbf{w}^*) &\leq \left\|\mathbf{w}^t - \mathbf{w}^*\right\|_2 \left\|\nabla f(\mathbf{w}^t)\right\|_2 \\ &\leq \left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2 \left\|\nabla f(\mathbf{w}^t)\right\|_2 \\ -\left\|\nabla f(\mathbf{w}^t)\right\|_2 &\leq -\frac{f(\mathbf{w}^t) - f(\mathbf{w}^*)}{\left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2} \end{aligned} \tag{3.50}$$

The last block to complete involves finding a relationship between successive iteration steps. Applying the Lipschitz property (3.14) between the points $\mathbf{w}^t$ and $\mathbf{w}^{t+1}$ together with equation (3.50) yields:

$$\begin{aligned} f(\mathbf{w}^{t+1}) &\leq f(\mathbf{w}^t) + (\mathbf{w}^{t+1} - \mathbf{w}^t)^T \nabla f(\mathbf{w}^t) + \frac{L}{2}\left\|\mathbf{w}^{t+1} - \mathbf{w}^t\right\|_2^2 \\ f(\mathbf{w}^{t+1}) &\leq f(\mathbf{w}^t) - \left(\eta - \eta^2\frac{L}{2}\right)\left\|\nabla f(\mathbf{w}^t)\right\|_2^2 \\ &\leq f(\mathbf{w}^t) - \left(\eta - \eta^2\frac{L}{2}\right)\frac{\left(f(\mathbf{w}^t) - f(\mathbf{w}^*)\right)^2}{\left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2^2} \\ f(\mathbf{w}^{t+1}) - f(\mathbf{w}^*) &\overset{+0}{\leq} f(\mathbf{w}^t) - f(\mathbf{w}^*) - \left(\eta - \eta^2\frac{L}{2}\right)\frac{\left(f(\mathbf{w}^t) - f(\mathbf{w}^*)\right)^2}{\left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2^2} \end{aligned} \tag{3.51}$$

In the last step the fact was used that $\eta - \eta^2\frac{L}{2} > 0$. To simplify the expression δ_t is introduced as $\delta_t = f(\mathbf{w}^t) - f(\mathbf{w}^*)$. Due to the definition of $\mathbf{w}^*$ as a minimum all δ in the sequence have to be positive. It is also known that δ_t is a decreasing sequence in t. To show this it suffices to compare the left- and right- hand side of the equation (3.51).

$$\begin{aligned} \delta_{t+1} &\leq \delta_t + \left(\eta^2\frac{L}{2} - \eta\right)\frac{\delta_t^2}{\left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2^2} \\ \frac{1}{\delta_t} - \frac{1}{\delta_{t+1}} &\leq \left(\eta^2\frac{L}{2} - \eta\right)\frac{\delta_t}{\delta_{t+1}\left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2^2} \\ \frac{1}{\delta_{t+1}} - \frac{1}{\delta_t} &\geq \left(\eta - \eta^2\frac{L}{2}\right)\frac{\delta_t}{\delta_{t+1}\left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2^2} \\ \frac{1}{\delta_{t+1}} - \frac{1}{\delta_t} &\geq \left(1 - \eta\frac{L}{2}\right)\frac{\eta}{\left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2^2} \end{aligned} \tag{3.52}$$

In the last step the fact was used that $\delta_t \geq \delta_{t+1}$. Finally, computing the sum over all the steps until $n-1$:

$$\sum_{n=1}^{t-1} \frac{1}{\delta_{n+1}} - \frac{1}{\delta_n} = \frac{1}{\delta_t} - \frac{1}{\delta_1} = \left(1 - \eta\frac{L}{2}\right) \frac{\eta(t-1)}{\|\mathbf{w}^1 - \mathbf{w}^*\|_2^2} \leq \frac{1}{\delta_t}$$

$$\frac{1}{\delta_t} \geq \left(1 - \eta\frac{L}{2}\right) \frac{\eta(t-1)}{\|\mathbf{w}^1 - \mathbf{w}^*\|_2^2} \tag{3.53}$$

$$f(\mathbf{w}^t) - f(\mathbf{w}^*) \leq \left(\eta - \eta^2\frac{L}{2}\right)^{-1} \frac{\|\mathbf{w}^1 - \mathbf{w}^*\|_2^2}{t-1}$$

The step size may now be taken as L^{-1} to tighten the upper bound.

$$f(\mathbf{w}^t) - f(\mathbf{w}^*) \overset{\eta=L^{-1}}{\leq} \frac{2L\|\mathbf{w}^1 - \mathbf{w}^*\|_2^2}{t-1} \tag{3.54}$$

q.e.d

The proof guarantees a decay in the function values while iterating which under the given assumptions translates into a shrinking error $\|\mathbf{w}^t - \mathbf{w}^*\|$. This however does not provide concrete information about the speed of the decay. From the right-hand side of the result, it can be also expected that GD has a high initial convergence that slows down when close to the optimum. This delay in convergence is not forcibly inconvenient in machine learning applications. Lax solver tolerances are normally set for two main reasons: Data outliers and noise are often part of the data set and trying to fit them perfectly to the model that is being trained is counteractive. Secondly, there is a phenomenon called overfitting which occurs when a model can predict seen data in a very good way whilst performing poorly on unseen data. A common procedure is splitting the available data samples in $\mathcal{D}$ into two categories: training and testing data. The model receives the train set to learn its parameters. During the training procedure, the test set is then used to evaluate the model giving regular feedback to the user about the model's performance. Overfitting is here seen when the error on the training data is diminishing while the testing score (accuracy) stagnates or grows with each iteration.

It is important to keep in mind that the proof is shown for a function bound to strict conditions. In normal applications of machine learning, it is not possible to guarantee that the error topology is convex and L-smooth. Nevertheless, the proof is still to some extent of use. In the case where a function is locally Lipschitz and quasi-convex (as well as twice differentiable) the GD is still ensured to converge towards a local minimum following the steepest descent from the initialization point $\mathbf{w}^1$.

A better convergence speed can be shown under a tighter convexity rule, called strong convexity.

Definition 3 (Strong convexity)**.** The function $f : \mathcal{B} \to \mathbb{R}$ is called strongly convex in $\mathcal{B}$ if for $\forall \mathbf{w}, \mathbf{y} \in \mathcal{B}$ and $t \in [0, 1]$ there exists a positive constant $\xi \in \mathbb{R}_{>0}$ such that:

$$f(t\mathbf{w} + (1-t)\mathbf{y}) \leq tf(\mathbf{w}) + (1-t)f(\mathbf{y}) - \frac{\xi}{2}t(1-t)\left\|\mathbf{w} - \mathbf{y}\right\|_2^2 \tag{3.55}$$

If f is twice continuously differentiable $f \in C^2(\mathbb{R}^n)$, then the strong convexity condition is equivalent to:

$$\mathbf{w}^T \left(\nabla^2 f(\mathbf{y})\right) \mathbf{w} \geq \xi \left\|w\right\|_2^2 > 0 \tag{3.56}$$

This higher convergence phenomenon can often be recorded when applying regularization to the minimization, i.e. when an artificial penalization term is used to force down the parameters in $\mathbf{w}$:

$$L(\mathbf{w}) = \hat{L}(\mathbf{w}) + \lambda \left\|\mathbf{w}\right\|_p, \qquad p \geq 1, \quad \lambda \in \mathbb{R}_{\geq 0} \tag{3.57}$$

Here, $\hat{L}$ represents the chosen loss function and λ is the regularization factor. The norms that are commonly used are the L_1-norm ($p = 1$) or the L_2-norm ($p = 2$). The reason for the speed-up lays in the strengthening influence of the regularization term on the convexity property. To elaborate on this statement, consider for instance an L_2 regularization and a convex loss function $\hat{L}$. Then, by definition of convexity (3.28) the Hessian matrix $\nabla^2 f(\mathbf{w})$ is positive-semidefinite. If now the Hessian matrix of the regularization term $2\lambda I$ is added, the whole matrix becomes positive definite and therefore strongly convex:

$$\mathbf{x}^T \left(\nabla^2 f(\mathbf{w}) + 2\lambda I\right) \mathbf{x} > 0, \qquad \left\|\mathbf{x}\right\| \neq 0 \tag{3.58}$$

Theorem 5 (Improved convergence of GD)**.** *Let $f : \mathcal{B} \to \mathbb{R}$ be a differentiable function satisfying the conditions of strong-convexity, L-smoothness on $\mathcal{B}$. Further, let $\mathbf{w} \in \mathcal{B}$ be an initial guess and η be a real valued small scalar sufficing $\eta \in (0, L^{-1}]$. Then the iteration* (3.10) *for $t = 1, \ldots, n$ will converge towards the minimum at w^* following:*

$$\left\|\mathbf{w}^t - \mathbf{w}^*\right\|_2^2 \leq (1 - \eta\xi)^t \left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2^2 \tag{3.59}$$

Proof.

As shown in theorem 8 for the constants ξ and L the relationship holds that $0 < \xi \leq L$. The beginning of the proof resembles the first steps of the normal GD proof and starts by simplifying

the relationship between the error for each time step.

$$\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2 \overset{(3.10)}{=} \left\|\mathbf{w}^{t}-\mathbf{w}^*\right\|_2^2 - 2\eta(\mathbf{w}^t-\mathbf{w}^*)^T\nabla f(\mathbf{w}^t) + \eta^2\left\|\nabla f(\mathbf{w}^t)\right\| \tag{3.60}$$

Writing out the L-smoothness condition 3.26 for $\mathbf{w}^{t+1}$ and $\mathbf{w}^t$ yields:

$$\begin{aligned} f(\mathbf{w}^t-\eta\nabla f(\mathbf{w}^t)) &\leq f(\mathbf{w}^t) - \eta\left\|\nabla f(\mathbf{w}^t)\right\|_2^2 + \eta^2\frac{L}{2}\left\|\nabla f(\mathbf{w}^t)\right\|_2^2 \\ f(\mathbf{w}^*) - f(\mathbf{w}^t) &\leq (\eta^2\frac{L}{2}-\eta)\left\|\nabla f(\mathbf{w}^t)\right\|_2^2 \leq -\frac{1}{2L}\left\|\nabla f(\mathbf{w}^t)\right\|_2^2 \end{aligned} \tag{3.61}$$

Also, due to strong convexity (7), the dot product may be substituted with:

$$-(\mathbf{w}^t-\mathbf{w}^*)^T\nabla f(\mathbf{y}) \leq f(\mathbf{w}^*) - f(\mathbf{w}^t) - \frac{\xi}{2}\left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2 \tag{3.62}$$

Introducing both inequalities into (3.60) yields:

$$\begin{aligned} \left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2 &\overset{(3.61)+(3.62)}{\leq} (1-\xi\eta)\left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2 + 2\eta(\eta L-1)(f(\mathbf{w}^t)-f(\mathbf{w}^*)) \\ &\overset{\eta\in(0,L^{-1}]}{\leq} (1-\xi\eta)\left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2 \\ &\leq (1-\xi\eta)^t\left\|\mathbf{w}^1-\mathbf{w}^*\right\|_2^2 \end{aligned} \tag{3.63}$$

By the relationship of ξ and L, it is clear that the factor $\xi\eta$ may take values between 0 and 1 (non inclusive) and therefore, the whole expression will tend towards 0 when $t \to \infty$.

q.e.d

Although GD is a fairly robust and simple algorithm, it faces difficult challenges in modern AI applications. Oftentimes, the learning set contains a large number of training samples, which cannot be handled in one computation. The gradient ∇f with respect to the parameters $\mathbf{w}$ is calculated with the whole training set and is commonly referred to as the true gradient. Performing this computation can however cause a significant slowdown. Consider for example the case, where the dataset has to inevitability loaded segment-wise from the main hard drive into the CPU. The comparatively long loading cycles cause the CPU/GPU to have a long idle time between computations, rendering the whole process inefficient. Instead, it is desirable to segment the data and lower the cost of individual iterations. This thought introduces the concept of Stochastic Gradient Descend method (SGD), an adaptation of GD meant to stem the challenges posed by big data.

3.3 Stochastic Gradient Descend

The new approach, SGD, use the loss of a single data sample instead of considering the whole training set in hopes that the gradient descent step will still march towards the local minimum. This reduces the computation time of every single iteration but may require more iterations in total to converge. The choices of $\mathbf{x}$ are realizations of a stochastic process, which in turn give the algorithm its name. When proving convergence, it is necessary to reformulate the goal, namely minimizing the error's expectation over all samples $\mathbf{x} \in \mathcal{D}$ for i.i.d. sampled $\mathbf{x}$ with uniform probability.

$$\mathbf{w}^* = \arg\min_{\mathbf{w}\in\mathbb{R}^n} \mathbb{E}\left[f(\mathbf{w}, \mathbf{x})\right] \tag{3.64}$$

The error term relates back to the total loss in the following way:

$$f(\mathbf{w}) = \frac{1}{n}\sum_{d=1}^{n} f(\mathbf{w}, \mathbf{x}^d) = \mathbb{E}\left[f(\mathbf{w}, \mathbf{x})\right] \tag{3.65}$$

The iteration rule for SGD varies little from its original form in GD. At each time step, a data point is sampled for computation.

$$\begin{aligned} \mathbf{w}^{t+1} &= \mathbf{w}^t - \eta \nabla f(\mathbf{w}^t, \mathbf{x}^t) \\ P(X = \mathbf{x}^t) &\equiv \frac{1}{n} \end{aligned} \tag{3.66}$$

Here again, it is possible to show some convergence properties of the expectation. For the sake of demonstration, proof for the case of strong convexity and Lipschitz smoothness will be shown.

Theorem 6 (Convergence of SGD)**.** *Let $f : \mathbb{R}^n \to \mathcal{B}$ be a twice differentiable function satisfying the conditions of strong-convexity and L-smoothness on $\mathcal{B}$. Further, let $\mathbf{w} \in \mathcal{B}$ be an initial guess and η a real valued small scalar in the range $(0, L^{-1}]$. Additionally, it is assumed that $\mathbb{E}[\|\nabla f(\mathbf{w}, \mathbf{x})\|_2^2] \leq B^2$ for any $\mathbf{w}$. Then, the iteration* (3.10) *for $t = 1, \ldots, n$ will converge towards the minimum at $\mathbf{w}^*$ satisfying:*

$$\mathbb{E}\left[\left\|\mathbf{w}^{t+1} - \mathbf{w}^*\right\|_2^2\right] \leq (1 - \eta\xi)^{t+1} \left\|\mathbf{w}^1 - \mathbf{w}^*\right\|_2^2 + \frac{\eta B^2}{\xi} \tag{3.67}$$

Proof.

As in the previous proofs, the initial step is writing out the relationship between $\mathbf{w}^{t+1}$ and $\mathbf{w}^t$ given by the iterates of SGD.

$$\left\|\mathbf{w}^{t+1} - \mathbf{w}^*\right\|_2^2 = \left\|\mathbf{w}^t - \mathbf{w}^*\right\|_2^2 - 2\eta(\mathbf{w}^t - \mathbf{w}^*)^T \nabla f(\mathbf{w}^t, \mathbf{x}^t) + \eta^2 \left\|\nabla f(\mathbf{w}^t, \mathbf{x}^t)\right\|_2^2 \tag{3.68}$$

Now the expectation is taken conditioned on $\mathbf{w}^t$.

$$\begin{aligned}\mathbb{E}\left[\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2|\mathbf{w}^t\right] = \left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2 - 2\eta(\mathbf{w}^t-\mathbf{w}^*)^T\mathbb{E}\left[\nabla f(\mathbf{w}^t,\mathbf{x}^t)|\mathbf{w}^t\right] \\ +\eta^2\mathbb{E}\left[\left\|\nabla f(\mathbf{w}^t,\mathbf{x}^t)\right\|_2^2|\mathbf{w}^t\right]\end{aligned} \tag{3.69}$$

The loss function is unbiased with respect to $\mathbf{x}$. In the discrete case it is easy to show that the gradient, too, is unbiased.

$$\mathbb{E}[\nabla f(\mathbf{w}^t,\mathbf{x}^t)|\mathbf{w}^t] = \frac{1}{n}\sum_{d=1}^{n}\nabla f(\mathbf{w}^t,\mathbf{x}^d) = \nabla\left(\frac{1}{n}\sum_{d=1}^{n} f(\mathbf{w}^t,\mathbf{x}^d)\right) = \nabla f(\mathbf{w}^t) \tag{3.70}$$

Using both the unbiased property and the L_2 convergence:

$$\begin{aligned}\mathbb{E}\left[\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2|\mathbf{w}^t\right] &= \left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2 - 2\eta(\mathbf{w}^t-\mathbf{w}^*)^T\nabla f(\mathbf{w}^t) + \eta^2B^2 \\ &\overset{(2)}{\leq}(1-\xi\eta)\left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2 - 2\eta\left(f(\mathbf{w}^t)-f(\mathbf{w}^*)\right) + \eta^2B^2 \\ &\leq(1-\xi\eta)\left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2 + \eta^2B^2\end{aligned} \tag{3.71}$$

The term $f(\mathbf{w}^t)-f(\mathbf{w}^*)$, which is omitted midway in (3.71) is positive for any $\mathbf{w}^t$ and resembles the quick convergence of SGD during the initial stages. To wrap the proof up, the inequalities are followed up to the initial member as followed:

$$\begin{aligned}\mathbb{E}\left[\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2|\mathbf{w}^t\right] &\leq (1-\xi\eta)\left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2 + \eta^2B^2 \\ &= (1-\xi\eta)\mathbb{E}\left[\left\|\mathbf{w}^t-\mathbf{w}^*\right\|_2^2|\mathbf{w}^t\right] + \eta^2B^2 \\ &\leq (1-\xi\eta)^{t+1}\left\|\mathbf{w}^1-\mathbf{w}^*\right\|_2^2 + \eta^2B^2\sum_{i=0}^{t}(1-\xi\eta)^i \\ &= (1-\xi\eta)^{t+1}\left\|\mathbf{w}^1-\mathbf{w}^*\right\|_2^2 + \eta^2B^2\left(\frac{1-(1-\xi\eta)^{t+1}}{1-(1-\xi\eta)}\right) \\ &< (1-\xi\eta)^{t+1}\left\|\mathbf{w}^1-\mathbf{w}^*\right\|_2^2 + \frac{\eta B^2}{\xi}\end{aligned} \tag{3.72}$$

Applying one last time the unconditioned expectation operator onto both sides yields the desired result:

$$\begin{aligned}\mathbb{E}\left[\mathbb{E}\left[\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2|\mathbf{w}^t\right]\right] &= \mathbb{E}\left[\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2\right] \\ \mathbb{E}\left[\left\|\mathbf{w}^{t+1}-\mathbf{w}^*\right\|_2^2\right] &\leq (1-\xi\eta)^{t+1}\left\|\mathbf{w}^1-\mathbf{w}^*\right\|_2^2 + \frac{\eta B^2}{\xi}\end{aligned} \tag{3.73}$$

Note that by the tower property of expectation, the conditioning on $\mathbf{w}^t$ can be dropped. *q.e.d*

In contrast to GD, the convergence bound has the constant positive factor $\frac{\eta B^2}{\xi}$ which prevents the algorithm from reaching an optimum for large t. Therefore, the user may expect an initial progression towards an optimum, followed by a possibly directionless straying of $\mathbf{w}$ close to the optimum. This behaviour can be well seen when implementing a simple optimization problem with two trainable weights. Take a set of data points in a 2D plane belonging to two different classes, 0 and 1. The quest for the perceptron will be finding a separating function that can determine whether a point belongs to one or another class. The bias, in this case, will be omitted. For the training, the two presented algorithms GD and SGD will be used and briefly compared. In figure 3.4 one can see the training progression on the left plot and the point clouds in the right plot.

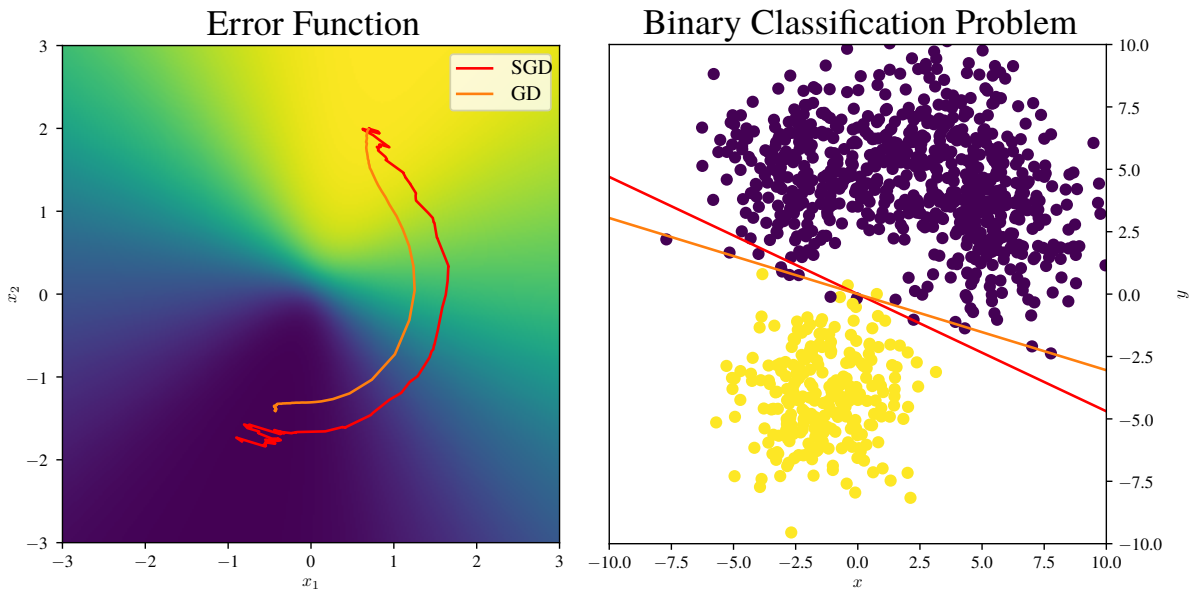

Figure 3.4: Perceptron training visualization comparing GD and SGD.

The colourmap of the background of the picture shows the relative error values depending on the choice of parameters x_1 and x_2. The perceptron was set up with the sigmoid function (3.5) that maps the linear combination of inputs onto values between 0 and 1. The algorithms were run with the maximal possible stepsize, in this case $\eta_{GD} = 2$ and $\eta_{SGD} = 0.35$ for a fixed amount of iterations $N = 100$. The two curves (red, orange) show how each of the mentioned algorithms adapts its steps parting from the same initial parameters. As can be seen, both algorithms follow similar initial directions towards the global minimum located in the southwest section of the figure. The convergence behaviour of GD is characterized by smooth, continuous

steps that progress towards the minimum. The stochastic algorithm on the other side tends to become increasingly chaotic when closing in on the minimum.

The performance of SGD in this scenario may be worse than the one of GD due to the simplicity and the small size of the problem. It may however be the case for other data sets that the optimization topology contains several local minima. In that case, the chaotic behaviour of SGD close to minima can be beneficial. GD will progress towards the closest local minima while SGD on the other hand side may reach the said local minimum, but can still jump out of it in one of its strays.

GD	SGD
+ Fast convergence per step	+ Small iteration cost
+ Big stepsize possible	+ Can overcome local minima
- Unsuited for large datasets	- Requires small stepsizes
- Converges to closest minima	- Memory loading overhead

Table 3.2: Overview of GD and SGD.

The two presented optimization algorithms have been shown to have their advantages and trade-offs in computation cost and convergence speed. It is natural to wonder whether a combination of both may yield good results: a Gradient Descend Algorithm using mini-batches of samples. This algorithm, oftentimes called Batch Gradient Descend method (BGD), is the most commonly used when dealing with Neural Networks. It offers the possibility of dividing the training set into even-sized batches, each of which is used for one gradient descend step. This method has a similar converge proof[13] as the previous algorithms and will be skipped for the sake of brevity. Its convergence properties transition depending on the chosen batch size. While for smaller batches the method will resemble SGD, bigger ones will turn out to be more closely related to GD. This allows the user to tune the batch size to fit the problem's requirements. For the remainder of the project work, this algorithm will be employed.

The presented methods have been refined and improved with small modifications such as the ones that will be shown in section 3.4. It remains however questionable if any other higher-order optimization techniques can be more efficient than the gradient methods. Most higher-order techniques require computing the Hessian matrix in order to use it in each update step. From a theoretical point of view, these algorithms enjoy a higher convergence rate and should provide valid approximations [14]. Unfortunately, computing the Hessian matrix is not viable for the common large scale ML problems. In section 3.5 the Backpropagation algorithm will be shown

and the difficulties in computing the gradient of the loss function for using it in SGD. If instead one was to compute the Hessian matrix of a given network, the computational overhead would drastically increase[14]. The attempts to alleviate this problem deal with finding a reasonable approximation of the Hessian. Sadly, they have not yet been studied and implemented efficiently enough for widespread use. For this reason and due to other inconveniences linked to constraints on the network architecture, higher-order solvers will not be considered.

3.4 Momentum

The presented stochastic optimization algorithms can be, and usually are, improved by using momentum terms. As the name may suggest, the gradient term and the step sizes in the descend iteration are modified considering past optimization steps. The new momentum steps aim to reduce the influence of noise while at the same time preserving the stochastic step's capability of overcoming local minima. The update term becomes now:

$$\begin{aligned} \mathbf{w}^{t+1} &= \mathbf{w}^t + \hat{\eta}^t g^t \\ g^t &= g^t(g^{t-1}, \nabla f(\mathbf{w}^t)) \\ \hat{\eta}^t &= \hat{\eta}^t(\eta, \hat{\eta}^{t-1}, g^t) \end{aligned} \tag{3.74}$$

Many distinct algorithms have been developed that treat the updates differently. Some, for instance, the Nesterov Momentum only perform changes on the gradient, while others like the commonly used Adam (Adaptive Moment Estimation) rule change both. In rough terms speaking, the newly introduced gradient g^t is a convex combination between the previous gradient g^{t-1} and the current gradient of the loss function $\nabla f(\mathbf{w}^t))$. It is hence often called the moving average or in more mathematical terms, the first-moment estimate. This causes the trajectory to be more consistent between iterations and less sensible to strongly altering gradients. Adam features also an adaptive step size that is also computed using the loss topology and the step sizes used in prior iterations. To be more precise, in Adam, the adaptive step size drops when the gradient's second moment or variance $(g^t)^2$ is large to avoid overshooting a large minimum[15]. According to testing benchmarks, [16], most first-order momentum algorithms consistently outperform the primitive versions of the gradient descend algorithms in terms of computation time and optimization results. What the best of those algorithms is depends from case to case and cannot be generalized. Lead by previous positive experiences gained at university studies and several positive reviews, the Adam algorithm will be used for the optimization in this project.

3.5 Neural Networks

So far, a simple learning technique has been shown and proven for the single perceptron. A single unit or even a layer of perceptrons, however, have quite limited use, since its capabilities are bound to the description of separating hyperplanes. With the desire to modelling complex relationships between the predictor values, a more versatile arrangement of perceptrons was brought up by mathematicians in the 1960's, the Multi-Layer Perceptron (MLP). The update to the previous model bases on adding additional, so-called hidden layers of perceptrons between the input nodes and the outputs. By using non-linear activation functions, the newly created network is capable of modelling far more complex functions. The MLP is the predecessor and one of the most simple versions of a feed-forward Artificial Neural Network (ANN).

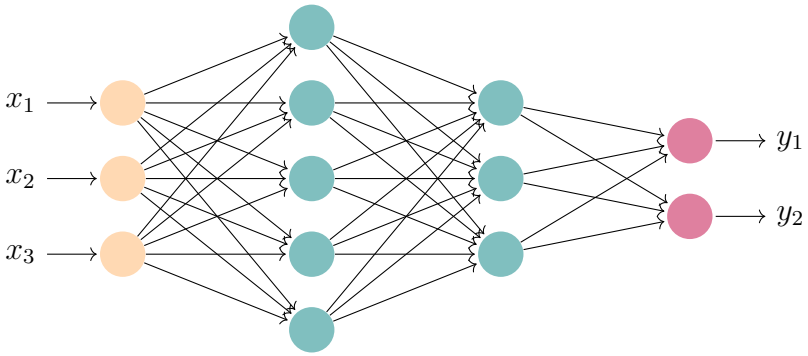

Figure 3.5: Densely connected ANN (5,3,2) with two hidden layers.

The keyword *feedforward* is given due to the information flow within the net that considering figure 3.5 means that input is passed from left to right. Neural networks that do not meet this criterion are for instance Recurrent Neural Networks (RNNs) since, inside them, the information may bounce back in the net several times before being output as a prediction. The human brain itself for instance is highly recurrent in its way of processing information [17].

The choice of the network depends on the type of problem, the expected complexity and feature space. Feedforward networks are usually employed when modelling problems that do not have a direct time or order dependency. Examples for this may be image classifications, geometry encoding or function approximations.

Feedforward networks offer the possibility to use different activation functions on each layer. The activation functions help to control and scale the values and gradients of processed inputs in the network. Generally, they are defined in a way that evaluating them is computationally cheap. It is often the case that the derivative can be expressed in terms of the activation function itself, saving additional resources. A small overview of the most commonly used activations is presented below to highlight some of their properties.

Sigmoid - The sigmoid function is continuously differentiable and non-linear in $\mathbb{R}$. It maps input values to the range $(0, 1)$ and is therefore often used at the final layer of classification networks to describe probabilities.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{3.75}$$

$$\sigma'(x) = \sigma(x)(1 - \sigma(x)) \tag{3.76}$$

Tanh - The Hyperbolic Tangent is in some sense similar to the sigmoid function. The smooth and nonlinear functions squashes values to the range $(-1, 1)$. It enjoys a slightly better convergence behaviour than the sigmoid function due to its larger derivative values. For this reason it is commonly used whenever the output of a inner layer has to be controlled, like in the case of recurrent units.

$$\tanh(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{3.77}$$

$$\tanh'(x) = 1 - \tanh^2(x) \tag{3.78}$$

ReLu - The Rectified Linear unit (ReLu) was created to tackle the Vanishing Gradient Problem that haunts other activation functions, like the ones described above. As will be seen later, a network's training involves terms that are multiplied with the derivatives of activations functions several times. These often yield values close to 0, downsizing the update steps with each evaluation. The ReLu function on the other hand is composed by combining two linear functions at a joint point. The most common implementation is:

$$\text{relu}(x) = \max(0, x) \tag{3.79}$$

$$\text{relu}'(x) = \begin{cases} 1 & x > 0 \\ 0 & \text{else} \end{cases} \tag{3.80}$$

The first aspect that might arise doubts is the origin ($x = 0$), where the function is clearly not differentiable. From a mathematical perspective this would break the convergence rules derived for the gradient descend methods. Despite the odds, the ReLu function works well and is in fact the most used one in deep neural networks like AlexNet or GoogLeNet. Depending on the implementation the gradient is set to either 0 or 1 at $x = 0$. Unfortunately, ReLu networks are prone to causing idle sections in the network. For a negative input sum, the function will return 0 and also have a 0 gradient. This in turn hinders the backpropagation rule, which will

be explained later from updating the upstream weights of that unit.

Apart from the activation functions it is important to consider different loss or error functions for neural networks in regression problems. The choice of error function defines how the optimizer will value the severity of the models mistakes. The first and most common loss function is the Mean Squared Error (MSE) that, as has been shown before, is defined over the average L_2 error of the prediction:

$$L(\mathbf{w}) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \|y_d - f(\mathbf{w}, \mathbf{x}_d)\|_2^2$$

Here, the neural network is summed up by the function f. The MSE is a common function in regression problems. The squared distances translate into large errors for samples that lay far away from the predicted values. In turn, the large error causes a strong correction of the predictor into that given direction. It is therefore suited for models that have to account for data outliers.

In addition to the L_2 based error there is also its L_1 counterpart, called the Mean Absolute Error (MAE). From the equations perspective, it is very close to the MSE with the only difference being that distances are measured using absolute values:

$$L(\mathbf{w}) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \|y_d - f(\mathbf{w}, \mathbf{x}_d)\|_1$$

The L_1 norm does not stretch out the error as much in the case of outliers, making it suited to handle data with noise in it. Overall, both functions provide solutions to regression problems and the difference in their performance is rather moderate.

Describing a neural network in mathematical terms requires an array of definitions for the multiple values involved in it. In the following, the index $k \in 1, \ldots, K$ will denote the index of a densely connected layer. Within each layer there will be a fixed number of neurons, each of them will receive the output from all the previous neurons. The connecting weights are denoted by w_{ij}^k, i.e. the weight connecting the i^{th} neuron of the preceding layer with the j^{th} neuron of the k^{th} layer. Then, the gathered inputs are passed through the activation function g^k which may vary between the different layers. Finally, the output of the neuron is denoted by o_j^k; the output of the j^{th} neuron in the k^{th} layer.

$\mathbf{x}$	Predictor sample
$\mathbf{y}$	Target output
w_{ij}^k, $\mathbf{w}_j^k$, $\mathbf{W}^k$	Weight of connection
g^k	Activation function
o_j^k, $\mathbf{o}^k$	Output value

Table 3.3: Quantities related to a neural network.

Note that by definition the outputs can also have the superscript 0, meaning $\mathbf{o}^0 = \mathbf{x}$. Using the presented notation one can formulate the information flow of a neural network with k layers as:

$$\mathbf{f}(\mathbf{x}) = g^k\left(\mathbf{W}^k \cdot \left(g^{k-1}\left(\mathbf{W}^{k-1} \cdot \ldots \left(g^1\left(\mathbf{W}^1 \cdot \mathbf{x}\right)\right)\right)\right)\right) \tag{3.81}$$

or equivalently in a recursive call with $\mathbf{f}(\mathbf{x}) = \mathbf{o}^K$:

$$\mathbf{o}^k = g^k(\mathbf{W}^k\mathbf{o}^{k-1}) \tag{3.82}$$

As in the case of the perceptron, the optimization target is finding a value for each weight in the net such that the error is minimized over the training set. Then, reusing the previously employed gradient descend rules, one can formulate the iterative weight update as:

$$w_{ij}^k = w_{ij}^k - \eta\frac{\partial L}{\partial w_{ij}^k}, \qquad \forall\, i, j, k \tag{3.83}$$

Here, since we are considering a case with several outputs, the error will be defined as the sum of errors for each of the targets. Computing the gradient of the loss function with respect to a weight in a multilayered network requires more dedication than in the case of the perceptron. As a starting point, the derivative can be expanded using the chain rule:

$$\frac{\partial L}{\partial w_{ij}^k} = \frac{\partial L}{\partial o_j^k}\frac{\partial o_j^k}{\partial w_{ij}^k} = \frac{\partial L}{\partial o_j^k}\frac{\partial o_j^k}{\partial \mathbf{w}_j^k \cdot \mathbf{o}^{k-1}}\frac{\partial \mathbf{w}_j^k \cdot \mathbf{o}^{k-1}}{\partial w_{ij}^k} \tag{3.84}$$

In literature, the term $\mathbf{w}_j^k \cdot \mathbf{o}^{k-1}$ is often denoted by net_j^k. This expansion is motivated by the chain of dependencies of global error. The first expansion stands for the relationship of the error with the output at one neuron in the net. The second expansion brings the choice of activation function into play. This can be better seen when expanding each partial derivative:

$$\frac{\partial \mathbf{w}_j^k \cdot \mathbf{o}^{k-1}}{\partial w_{ij}^k} = o_i^{k-1} \tag{3.85}$$

This term scales the learning rate of the weights with its input value. For this reason, it is important not to initiate the weights in a way that many outputs result in small magnitude numbers. Then, the second term, as mentioned before, brings in the direct relationship between the outputs and the activation functions:

$$\frac{\partial o_j^k}{\partial \mathbf{w}_j^k \cdot \mathbf{o}^{k-1}} = \frac{\partial g^k(\mathbf{w}_j^k \cdot \mathbf{o}^{k-1})}{\partial \mathbf{w}_j^k \cdot \mathbf{o}^{k-1}} = g'^k(\mathbf{w}_j^k \cdot \mathbf{o}^{k-1}) \tag{3.86}$$

This term scales the update steps with the gradient of the activation function itself. This term is again relevant when initializing the weights of a neural network or scaling the predictor before feeding it into the network. Suppose for instance that a neural network is set up with the sigmoid function in each layer. Further, assume that the weights are initialized in a way that multiplying them by the inputs yields outputs with large magnitudes. Then, due to the small values of the derivative of the sigmoid function the update steps will be small. This in turn results in a slow learning rate of the affected neurons.

The remaining term is computationally seen as the most expensive and difficult to calculate. The derivative of the error with respect to one particular weight w_{ij}^k may unfold over a large number of computations. The error gradient is treated in the same sense in layers excepting the last one, where $k = K$. The error is then directly dependent on the weight that is optimized and no further computations are needed. For all other $k \neq K$ it is necessary to expand the terms for each layer located between the weight and the output layer. Recall that the error is a sum of the losses of the final outputs. With some abuse of notation, one may describe it as followed:

$$\begin{aligned} L := L(\mathbf{y}, \mathbf{o}^K) &= \sum_j \hat{L}(y_j, o_j^K) \\ \mathbf{o}^K &:= \mathbf{o}^K(o_j^k) \end{aligned} \tag{3.87}$$

$$\begin{aligned} \frac{\partial L}{\partial o_j^k} &= \frac{\partial L}{\partial \mathbf{o}^K} \cdot \frac{\partial \mathbf{o}^K}{\partial o_j^k} = \frac{\partial L}{\partial \mathbf{o}^K} \cdot \frac{\partial g^K(\mathbf{W}^k \mathbf{o}^{K-1})}{\partial o_j^k} \\ &= \frac{\partial L}{\partial \mathbf{o}^K} \cdot g'^K(\mathbf{W}^k \mathbf{o}^{K-1}) \mathbf{W}^K \frac{\partial \mathbf{o}^{K-1}}{\partial o_j^k} \end{aligned} \tag{3.88}$$

The terms in (3.88) may as well be described in a more fashionable way using a recursion:

$$\frac{\partial \mathbf{o}^k}{\partial o_j^l} = \begin{cases} g'^k(\mathbf{W}^k \mathbf{o}^{k-1}) \mathbf{W}^k \frac{\partial \mathbf{o}^{k-1}}{\partial o_j^l} & \text{if } k > l \\ \mathbf{e}_j & \text{if } k = l \end{cases} \qquad k \geq l \geq 1 \tag{3.89}$$

An optimal way of computing the backpropagation rule is starting from the deepest layers and storing the computed loss derivatives for the whole network (backpass). Then, once done with that, the weights can be updated starting from the input layer towards the output using the so-called cumulative gradient from the backpass. Although the derivation shows the update of a sole weight, it is normally done for all weights in a layer simultaneously using higher-order tensor operations. This connection to tensor products is what gives for instance Google's API Tensorflow its name. It is important to note that the algorithm may also be applied with little modifications to other types of layers e.g. convolutional layers (section 3.6).

The Backpropagation algorithm has regardless of its versatility also some downsides. Numerous products with evaluations of activation functions frequently result in small updates of the weights far upstream. In the case of Deep Neural Networks (DNNs) with smooth activation functions, i.e. networks with a large number of hidden layers [17], the weights in the initial layers often have to be pre-trained, since they otherwise barely change during training. This phenomenon is referred to as the Vanishing Gradient Problem in the machine learning community. As the name says, the gradient used to update the weights tends to become smaller with each backward step in the neural network. It can be seen from equation (3.89) that for each level away from the outputs the gradient is multiplied with the derivative of the activation function. The derivatives of both the sigmoid function and the hyperbolic tangent are strictly bound by $\pm 1/2$. Therefore, each additional level of depth in the network downscales the update. As stated before, activation functions like the ReLu attempt to solve this issue. It holds that its gradient does not diminish as strongly since $\text{relu}' \in \{0, 1\}$. It is however the unfortunate case that another problem is related instead with this kind of function. Since the gradient is indeed 0 for negative values that reach the activation function, a neuron may be never updated. In a large network, one can expect this phenomenon to strike several neurons at a time, causing a considerable loss of functionality. The ML community, in its creativity for naming, refers to this issue as the Dying ReLU Problem. Although some more complex versions of the ReLu have been brought into existence, there is no clear evidence of them performing significantly better [18].

3.6 Convolutional Neural Networks

So far, the only type of layers considered were densely connected ones. These, far from being the only ones used, are rather unsuited to handle images alone. Image and volume data (3D images) generally provide a large number of pixels for a neural network to process. Imagine having a dataset of images with the size 256 by 256 pixels that ought to be classified. The flattened input would have 65536 entries. If it is now passed onto a small-sized dense layer with 100 neurons, the net would already have 6553600 trainable weights; an extraordinary number of parameters for only one layer. In face of this issue, it becomes clear that other processing techniques are required to handle this kind of data. Convolutional Neural Networks (CNNs) offer a solution: their architecture includes two new kinds of layers that deflate the number of parameters while preserving most of the information.

The first and most important of these layers are convolution layers. These layers perform local discrete convolutions of trainable kernels with the image data to extract local features. Convolutions as filtering tools have a long history in image processing. An example is the Laplacian filter for edge detection, a convolution using a so-called laplacian kernel:

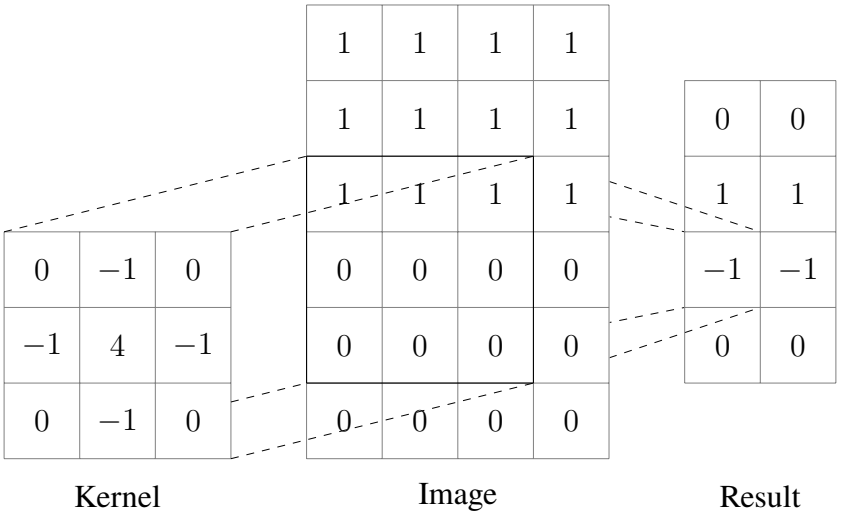

Figure 3.6: Convolution with 3 by 3 laplacian kernel without padding.

The image shown in figure 3.6 is processed by the convolution into a smaller representation. The resulting image has 0's on fields that were either fully populated by 0's or 1's in the original image. Boundary points on the contrary have numbers distinct from 0. Executing such a convolution on larger images with laplacian-like kernels results in images like the one shown in 3.7. The output image shows enhanced edges of the original image.

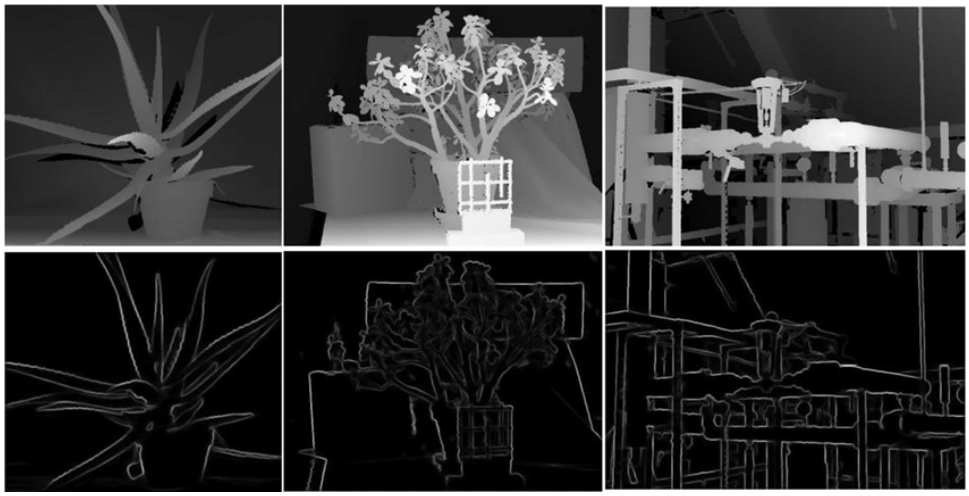

Figure 3.7: Gray colour images (above) and the result of a laplacian convolution (below) in absolute values [19].

Convolution layers in neuronal networks base on the same computing principle but have trainable weights as kernel entries instead of fixed values. The reasoning behind this is that it is now the network's task to find meaningful patterns in the image that yield information for a possible regression or classification problem. Compared to dense layers, convolutional layers do not consider the whole image at the same time, but rather local segments of it. This helps to reduce the number of weights to train without significantly reducing the prediction capability [20]. Convolution layers generally process geometrical data using two or three-dimensional kernels while one-dimensional kernels may also be used to process time-series data. Convolution layers do not only process data with one channel as shown in figure 3.6, they also deal with multi-channel inputs such as RGB images or previous outputs of convolution layers. Here, the information in one region is processed by one neuron for all present channels simultaneously. One can see this as a dense network in the image depth.

The general way in which convolutional layers in a network are defined is by providing a kernel size (usually odd-sized rectangular kernels) and a number of output channels. The output channels define how many different features ought to be abstracted from the input for one region. One can better see this on the 1D convolution depicted in figure 3.8. There is a set of 5 inputs in convolved using a 1D kernel with size three and two output channels. Additionally, parameters like the stride and the padding can be also defined. The stride is an integer that defines the spatial offset between convolutions. In figure 3.6 the stride was set as to be one since the kernel moves by one unit each time a different convolution is calculated. The padding is a conceptual argument that defines the convolutions behaviour at the boundaries of the image. One option that is quite popular for image classification is using the valid padding, i.e. only performing

convolutions on fully defined data (see figure 3.6). This approach is computationally wise the most effective since it downsizes the image size in each layer where it is used. While designing the network it is important to pay attention to the dimensions of the hidden states to ensure adequate data handling. The other options do not modify the input size and thus make the design process simpler. Layers of values are added around the image such that the convolution results in the same size as the original. The padded values are usually set to 0, but can also be mirrored values of the boundary.

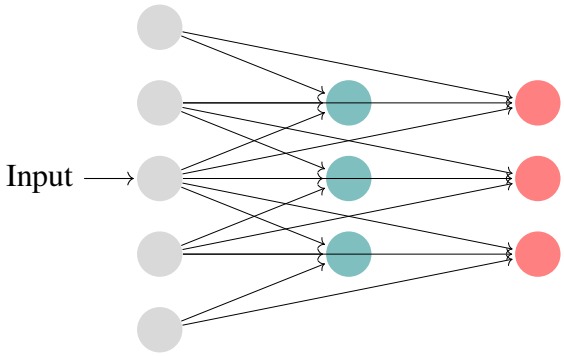

Figure 3.8: Schematic arrangement of a 1D convolution neuron with two output channels.

The neuron's inputs are treated for the rest in the same way as the ones of a dense layer. The sum of weighted inputs is then added up with a trainable bias and processed by an activation function. It is then common to chain up several convolution layers that progressively pass information to neighbouring regions of the hidden states (outputs of hidden layers).

Pooling layers are the second player in CNNs and, contrary to the convolution layers, do not have trainable weights. The pooling operations boil down to downsampling the data locally following a maximum or average rule. Commonly, pooling regions have a small, even size, e.g. length 2 in each direction. This already implies halving the size of the input for each dimension. Maximum value pooling, also called max-pooling, has a natural synergy with ReLu activated convolution layers. The pooling promotes non-zero (positive) values to be passed through the network, minimizing the repercussions of the Dying Relu Problem. Another advantage that it brings is helping to deal with input translations. Suppose for instance a picture is fed into a convolution layer and pooling layer. If that images information is shifted by one position, the pooling will return a completely different picture than in the non-shifted case regardless of the convolution layers before it. By this, it does prevent convolution layers of fixing features to only one section of the image. This is particularly important when few training samples are available and the training set has to be manually enhanced with image translations and dilations.

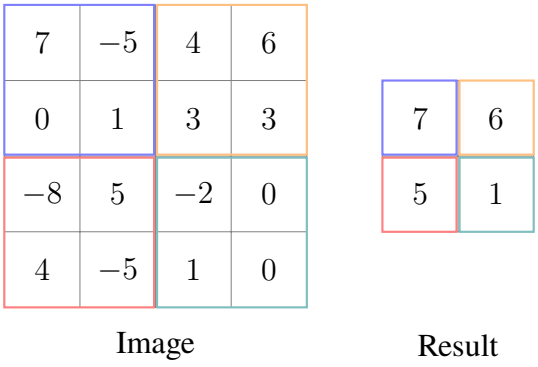

Figure 3.9: Pooling operation for a 2 by 2 region with a stride of 1 (skipping one position).

CNNs are generally built by cascading image information through convolution layers and max-pooling layers until it reaches a small enough size. Then the hidden state is flattened into a 1D array and fed to dense layers. The dense layers compute the correlations of the image features and process the data into a shape set by the user. Other networks as the U-Net that is employed for image segmentation tasks, skip dense layers and perform convolution, max-pooling and up-sampling steps only.

Chapter 4

Model

The current section will combine the knowledge of neural networks with the physical foundations of foaming simulations to assemble a workflow ranging from the generation of simulations to the evaluation of the prediction network. The set goal of the optimization is to assemble a program that takes in a surface mesh and outputs a heatmap with predicted simulation results for a range of possible input combinations. The model will be provided with data bundles of geometries and simulation setups and then trained to predict the corresponding simulation outcomes. These outcomes are quantities computed at the end of a simulation that can be directly correlated with a good or bad choice of parameters. The network should then be able to provide quick predictions on how good the given sets of parameters are for a geometry. This information can then be either used visually or numerically to optimize the production parameters. In the following section a concept for the problem at hand will be provided answering the key questions:

1. What is the relevant information to feed the network?
2. How are the different data sources preprocessed and scaled?
3. What are the targets for the network?
4. What is a good architecture for the neural network?

4.1 Voxelization

The moulds geometry is key information to be used. In an ideal case, the neural network should recognize the type of geometry based on its features and relate it to previously seen data. This visual correlation is one of the reasons for using neural networks in the first place since few other techniques in ML can apprehend geometrical features. For the data to be used it is necessary to preprocess it into a uniform shape that allows spacial correlations to be extracted from it.

The STL format is a simple and widespread convention for 3D surface meshes in which the geometry's surface is described using non-overlapping triangles [21]. The files are composed of a list of vertices and surface normal vectors and can be found both as ASCII files or binary files. Many CAD programs and post-processing tools like Paraview [22] have native support for these files. For this considerable advantage incompatibility and structure, this format is chosen for the project. To generate the needed shapes the parametric cad tool Jscad [23] has been used. It provides a simple tool to assemble STL files using javascript scripts online. The used training set contains a series of 85 distinct geometries, all of which are constant along the z-axis and have the same thickness. All of the files fit in a box of the dimensions (in meters):

$$x \in [-0.2, 0.2], \qquad y \in [-0.2, 0.2], \qquad z \in [-0.01, 0.01]$$

Generating surface files with an exact number of data points is without a question unfeasible, but fixed-sized inputs are required by feed-forward networks to function. In face of this issue, it is necessary to transform a generic surface mesh into a fixed size mesh. This can be achieved by voxelization, i.e. the conversion into a 3D volume mesh with rectangular cells. Voxel meshes are rudimentary compared to the widely used tetrahedron meshes. Resolving rounded surfaces with rectangular elements is, as one may expect, suboptimal and requires high cell densities. On the counter side, they can be easily generated and scaled thanks to the cells' regular structure.

The meshing procedure starts with reading in the STL file and defining its bounding box, i.e. the coordinates of a rectangular box in which the whole geometry may fit. To help simplify debugging the code one can scale the STL file to a size where its largest dimension has size 1. Then, the program proceeds by iterating through each of the xy-planes in the empty voxel cube. The procedure on each plane is then:

1. Collect all the triangles that cross or touch the xy-plane based on the vertices z-coordinate.
2. Compute the intersections between the triangle edges and the plane and store the intersections pairwise for each line pair of a triangle.
3. In the case of a triangle touching the plane with some of its vertices the coordinates of those vertices may be stored directly without computing any intersection.

4. Iterate through each x-line in the plane and calculate possible intersections with the lines spanned by the points in 2.

5. Following the sandwich principle, the corresponding lines in the voxel cube are populated with 0's and 1's depending on the order in which the x-line is intersected.

The presented algorithm that has been presented has been implemented in python. A sample result can be seen in figure 4.1.

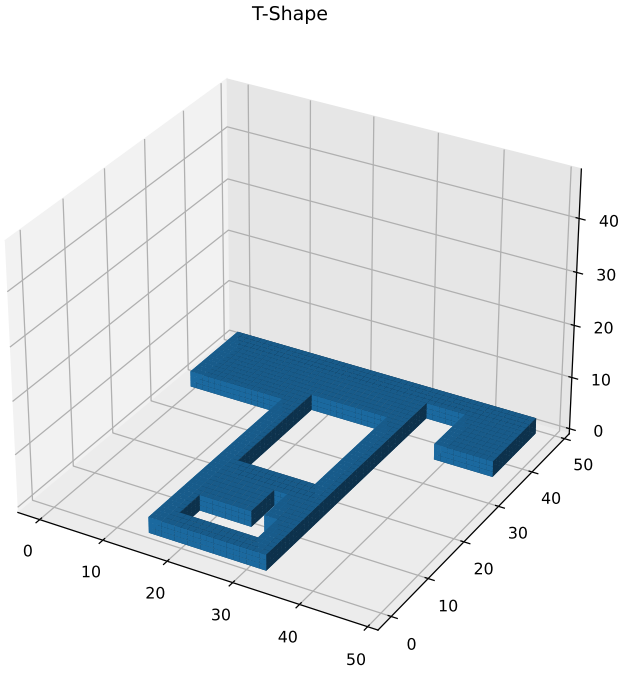

Figure 4.1: Results of voxelization.

The other geometries are also things along the z-axis and have a constant cross-section. This has been chosen in that way to simplify the data sampling (following section) and to reduce the model size. Since there is no evidence or proof for such a model to work, it is deemed more important to see whether a simplified quasi-2D model succeeds in its predictions. The invariance of the geometries along the z-axis allows it to take only one slice of the voxel mesh for the assembly of the predictor instead of a full 3D cube.

4.2 Data sampling

The next topic to deal with is the choice and sampling of the simulation parameters that ought to be optimized. Naturally, the foaming process carries many settings to improve the outcome. The most notorious ones are the injection paths, injection speeds and tilting the geometry for a better polymer distribution in its inside. Due to long-lasting simulation cycles in the data generation, not all of these parameters will be considered. Note that each additional parameter is likely to require a multiple of the simulations to be done for it to be predicted correctly. The ones that will be used are:

- Injection location $[x, y]$
- Mould rotation angle $\theta \in [0, 2\pi)$

Every simulation case will therefore be executed for one pair of input coordinates and a certain pitch angle. The pitch angle defines the degree of inclination of the geometry around the z-axis. The gravity vector, which is used to model the tilt, is defined in the xy plane and starts pointing in negative y-direction for the angle $\theta = 0$. The training cases are set up for a total of eight angles, starting from $\theta = 0$ up to $\theta = 7\pi/4$ increasing in $\pi/4$ steps. To reduce overfitting, these angles have a small random deviation in the range of $\pm\pi/30$ (± 6 degrees).

Finding plausible injection locations on the other side can be more difficult. Randomly sampled locations can lead to erroneous simulations and therefore either cause computational overhead or bring up outliers in the training set. A remedy to this is marking surfaces in the STL files as input faces and sampling points in those areas. The meshing tools used for the voxelization can partially be reused by taking the triangles in the inlet faces and dividing them based on their location and orientation. Each of those inlets is then parametrized based on the surfaces corner points and a minimal distance toward sharp edges. The inputs can then be sampled in a user-defined quantity over all the faces of the geometry. Here again, a random small offset is added onto the x and y coordinates of the injection location to avoid the same values from being used over and over.

The training set is then assembled by computing a series of simulations and progressively gathering the results. The set comprehends, as stated before a total of 85 geometries for training. Each of the geometries is computed for 14 different injection locations and 8 angles. The resulting dataset has a total of 9520 samples used for training. Additionally, 5 other geometries are used for validation purposes, each of them being evaluated at 25 locations and 8 angles without deviation.

In the derivation of the backpropagation, it has been shown that the inputs should be scaled to avoid slow initial convergence. The injection locations for instance should ideally be defined

in the range $[-1, 1]$ based on the dimensions of the bounding box. This simplifies the correlation between the geometry input and the injection location since no extra information about the geometries dimension has to be given. The angle of tilt is also scaled to the unit interval. The geometry on the other side is already given as an array of 0's and 1's and does not require scaling. Test runs with and without a linear scaling of the geometry have shown no particular differences in convergence speed or accuracy.

4.3 Network Targets

The choice of simulation outcomes is debatable from a physical perspective. A good foaming procedure yields a component that fills the mould and has an isotropic foam density. A partial filling shows clear flaws and is discarded immediately. The homogeneous foam material is often needed to guarantee the strength and resistance of the material. To control both criteria three outcomes are outlined:

1. Fillratio:
$$\delta = \frac{\int_\Omega \alpha \, dV}{\int_\Omega dV} \tag{4.1}$$
2. Maximum foam density:
$$\rho_{max} = \max(\rho_F) \tag{4.2}$$
3. Density deviation:
$$\rho_{dev} = \frac{\max(\rho_F) - \int_\Omega \rho_F dV}{\max(\rho_F) + \int_\Omega \rho_F dV} \tag{4.3}$$

Figure 4.2 shows two identical geometries being filled from different positions. The left figure is filled from the bottleneck in the upper right part of the T-shaped geometry. Large air encasings can be seen at the top left corner and the lower sections of the geometry, where the foam streams meet at the end of the expansion process. These air bubbles stand mayor production flaws and also compromise the component's material strength. The right simulation on the other hand, has its injection point at the bottom right corner and shows better results for both fill ratio and maximal density. The core of the mould is here properly filled and only shows smaller air cavities in the upper edges. Although not ideal, this second simulation yields better results than one on the left picture. The only output that does not agree with this statement is the density deviation, which is smaller in the left simulation. The wave-like patterns are caused in both simulations by trapped air that remains in the mould. The foam expansion drags these mostly small air bubbles causing a drop in the density profile at those locations. Small density drops are usually

only a phenomenon that is seen in simulations, while larger ones can be seen in experiments too.

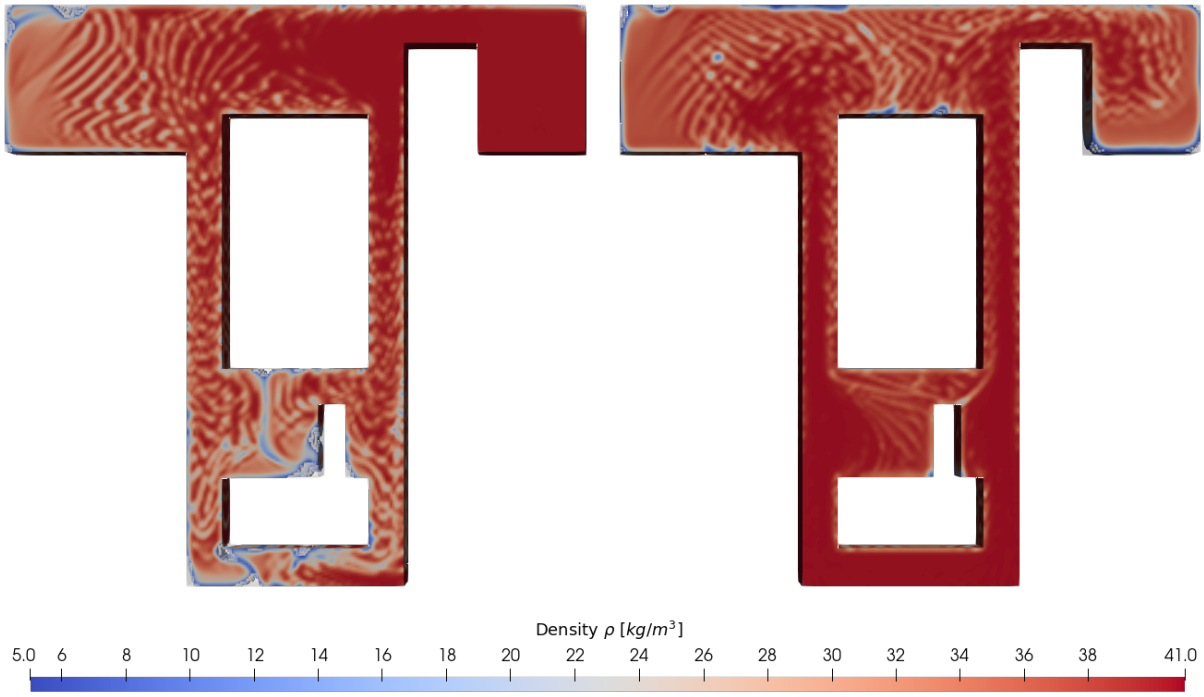

Figure 4.2: Comparison between a bad setup (left) and a good one (right).

Quanitity	Left	Right
δ	0.934	0.939
ρ_{max}	$40.82 kg/m^3$	$40.80 kg/m^3$
ρ_{dev}	$1.91 \cdot 10^{-3}$	$1.95 \cdot 10^{-3}$

Table 4.1: Simulation results for the simulations shown in figure 4.2.

The system of equations deals with a material model for the foam's density that performs strong updates on the expansion rate of the fluid. In some rare cases, these updates can cause a portion of the polymer mass to be lost. This is most likely caused by a numerical issue in the simulations on coarse grids. It is therefore important to keep track of the mass conservation to ensure that the gathered results are physical and not influenced by numerical errors. A new quantity, the mass conservation ratio $\hat{m}$, will also be evaluated and checked at the end of each simulation:

$$\hat{m} = \frac{\int_\Omega \rho_F dV}{\int_{t=0}^{T} \left(\oint_{A_{inlet}} \rho_F \mathbf{v} \cdot \mathbf{n} \, dA \right) dt} \tag{4.4}$$

The conservation ratio roughly speaking compares the polymer mass flux $\rho_F \mathbf{v}$ entering the mould through the inlet A_{inlet} with the foam mass present at the end of the simulation. Only simulations with a ratio larger than 95% are considered for training and validation.

Lastly, it will be necessary to apply a standard scaling to the outputs too. Although scaling is normally only applied to the inputs and is uncommon for the outputs it is necessary in this case. All three targets are assumed to be equally important for the optimization. The errors of each of the outputs ought to be seen in context with the general magnitude of that variable: $\delta \approx [0.8, 1]$, $\rho_{F,max} \approx [30, 70]$, $\rho_{F,dev} \approx [0.001, 0.01]$. An error of 0.1 of the maximum density $\rho_{F,max}$ is negligible, while the same deviation would be unacceptable for the fill ratio δ. If the model was to predict unscaled targets, it would weigh all errors equally and would thus aim to match the maximal density at the cost of the other targets. The employed standard scaler simply subtracts the data's mean and divides the result by its variance. The scaling factors are stored after training and will be used whenever the model is evaluated.

4.4 Model architecture

The predictor is composed of two input streams, one being a set of binary cross-section images of the geometry and the other one being a set of vectors with three entries corresponding to the simulation setting. Both streams need to be processed simultaneously by the network but are handled differently. The geometries are fed into a set of ReLu- activated convolutional and pooling layers to compute a smaller abstract representation with spatial information. At the end of the geometry processing, the inputs are flattened into a 1D vector. At this stage, the simulation parameters, which have passed one dense layer themselves, are concatenated with the geometry information and forwarded to a larger dense layer. To avoid overfitting, this dense layer is additionally geared up with dropout regularization, a technique that randomly sets a portion of the outputs of the layer to 0 during training. The information is then passed onto one regular ReLu activated dense layer and subsequently to the last dense layer of size 3. This final layer forwards the sum of inputs without an additional activation function. An overview of the model is given in figure 4.3 in which each coloured block represents one active layer of the network. The proportions of each block are a rough representation of the layers output shape. Note that the shrinking of the image size in each layer is very much present although not represented in the scheme.

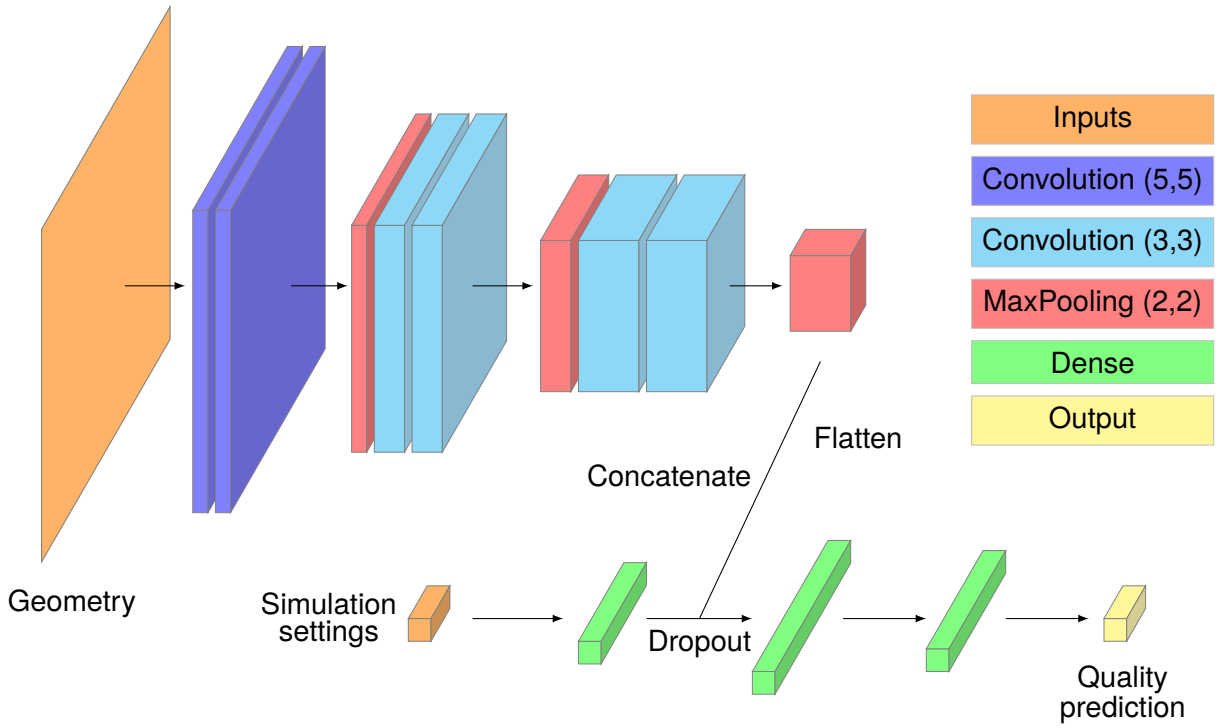

Figure 4.3: Schematic view of the used neural network.

The exact number of layers is motivated by two constraining factors: the complexity and its varying training efficiency and the image shape consistency. As seen in the previous section, the training speed of the last layers can be severely harmed by having too many layers. This issue can be seen easily when adding further layers to a network without a significant improvement in the testing score. The first layer's weights barely change and the rest of the network is simply trained on a non-linear conversion of the inputs instead. Therefore, an iterative addition of layers and comparison of the learning progression is needed to find an optimal structure. The different kernel sizes have also been set to favour better applicability of the net to other geometries. Due to having a downsizing of 4 units in each dimension of the image for the $(5, 5)$ kernels, it is achieved that the image size is always even in the network. It is not just the case for the used 48 by 48 images, but for a variety of other sizes like 40, 56, 64, etc. The detailed size progression can be seen further below in figure 4.7.

Not just the number of layers but also many other hyperparameters such as the number of neurons of each layer or the dropout rate have to be tuned. To find a good set of them, it is necessary to tune the model with several test runs. Based on a tuner's rule, suggestive parameters are tried out. Each time the model is run, the loss over a random 10% split of the training data is computed and set as the run's score. Due to the large size of the training set, no significant differences have been seen for different splits.

According to experimental tests[24],[25] concerning the different tuning algorithms such as Random Search, Hyperband, there is no clear best algorithm. Among the run tests, Random Search stood out as being one of the tuners with the most consistent performance, making it the algorithm of choice in this project. The array of tested options and scores can best be analyzed using parallel plots. The axes that part the graph vertically mark the values of the tested parameters for each run. The subset of parameters chosen is given in the enumeration below.

- `conv_units` : Factor to determine the number of channels in each convolution layer n_{conv}. Each pair of successive convolutions shares the number of channels. The number of channels between different blocks is multiplied by the factor 2.
- `dense_units2` : Number of neurons in the second dense layer n_2.
- `dense_units3` : Number of neurons in the second dense layer n_3.
- `dropout_rate` : Dropout rate $\hat{d}$.
- `learn_rate` : Learning rate η.

In figure 4.4 a total of 200 runs can be seen with the corresponding hyperparameters and test scores:

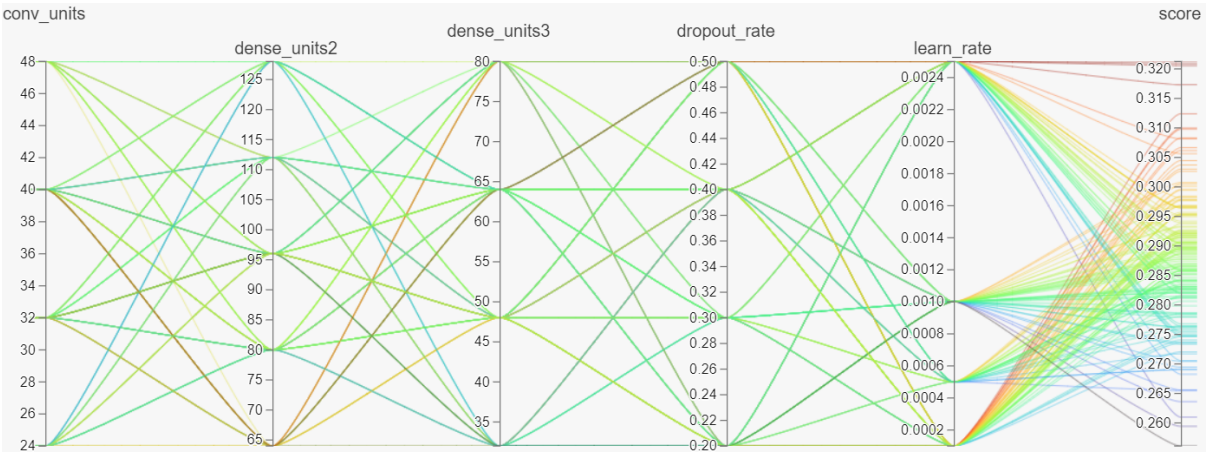

Figure 4.4: Parallel plot for some of the tuned parameters.

The last bar, *score*, corresponds to the loss over the test set computed after 5 epochs and defines the colouring of the lines. In the presented plot in figure 4.4 only a small subset of the fitted parameters is shown for visibility reasons. The chosen hyperparameters belong to the last iteration done to determine the final model. In a relatively chaotic graph, as 4.4, it can be difficult to

infer correct assumptions. A way of extracting information from it is by isolating curves based on single parameters or scores. This for instance can help to determine parameters that cause a high variance in performance or generally bad scores. A demonstration of this is shown in figure 4.5, where the fraction of the lines corresponding to bad scores is inspected.

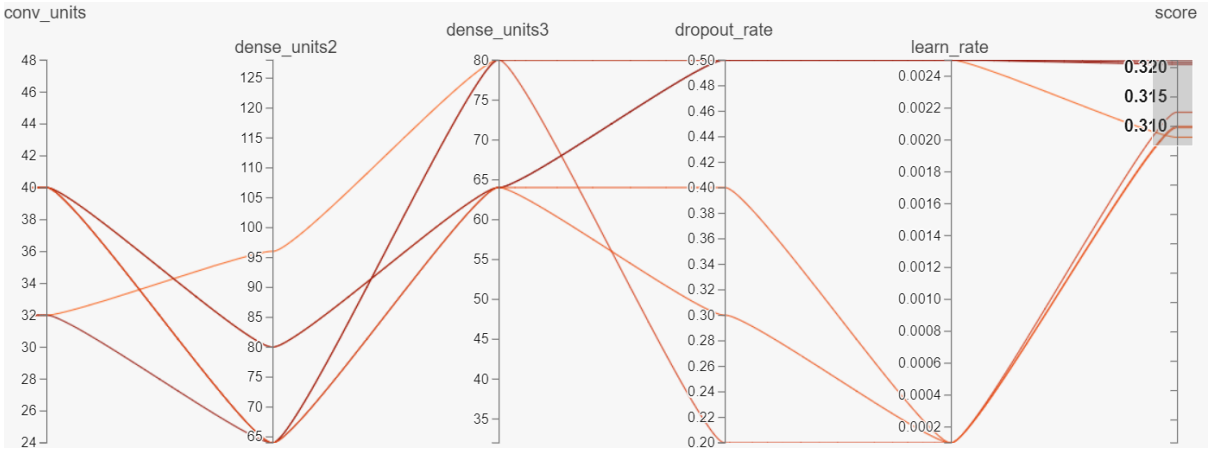

Figure 4.5: Parallel like 4.4 keeping bad scores.

It can be observed that the worst running cases follow the trend: Small numbers for n_2, large numbers for n_3 and either very large or very small learning rates η. The dropout is very distinct for all runs, so no clear relationship can be inferred from it. This scenario can then be compared to a plot isolating all good scores (see figure 4.6).

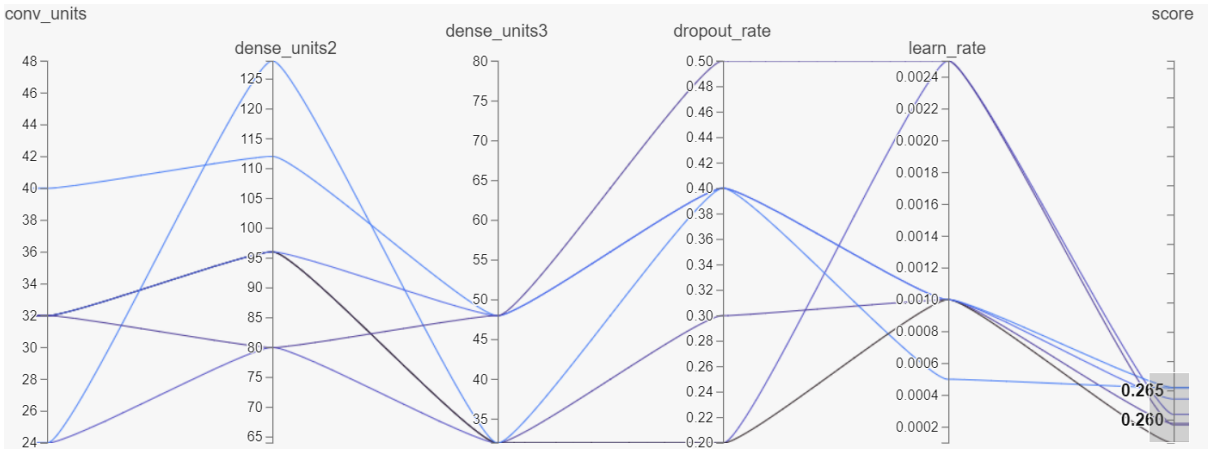

Figure 4.6: Parallel like 4.4 keeping good scores.

In this plot, good scores are generally obtained with a larger second dense layer and a smaller third one. The most productive learning rates dwell around $\eta = 0.001$, but also some good runs have been recorded for $\eta = 0.0025$. This large learning rate causes, when considering also figure 4.4, a high variance in the score. Other rates, such as $\eta = 0.001$ provide a consistent, good score range and are hence preferred. For the dropout, again, there is no visible trend for its values.

The parameter tuning has provided some guides for building the model, although not all parameters have been properly resolved. Whenever there is a margin of choice, for instance with the number of convolution channels n_{conv}, one should opt for a simple configuration following the law of Occam's Razor:*"Entities should not be multiplied beyond necessity"*[26]. This rule is not just adopted from the friar William of Ockham (14^{th} century), who started it, but also due to its implications in ML. For the first, training and evaluating a smaller neural network is quicker and requires less memory. Secondly, smaller networks generally help to prevent overfitting by offering less room for remembering exact data samples. The final choice of parameters can be seen in figure 4.7.

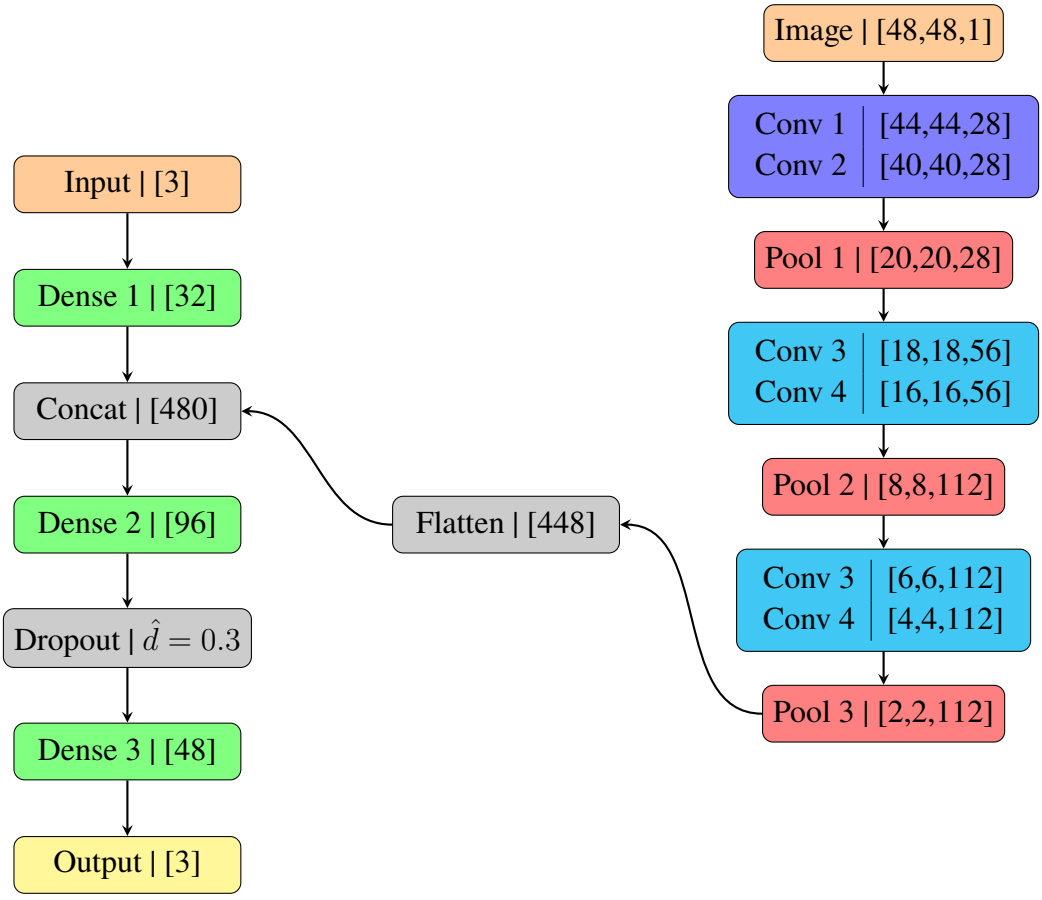

Figure 4.7: Model structure.

Chapter 5

Results

To model proposed in the previous section is then trained with a learning rate of $\eta = 0.001$ and a batch size of 4. The batch size is chosen to be small to profit from its better exploratory behaviour, as shown in the derivations of section 3. It is nevertheless not set to 1 to improve the computation speed and to enjoy a slightly faster convergence. During the training, an early stopping criterion is used, i.e. a simple callback stopping the training procedure, when the test score does not improve in some epochs. Under these conditions, the neural network takes roughly 10 minutes to train for a test loss of $L = 0.22$. The training progression together with the testing loss and the validation loss can be seen in figure 5.1:

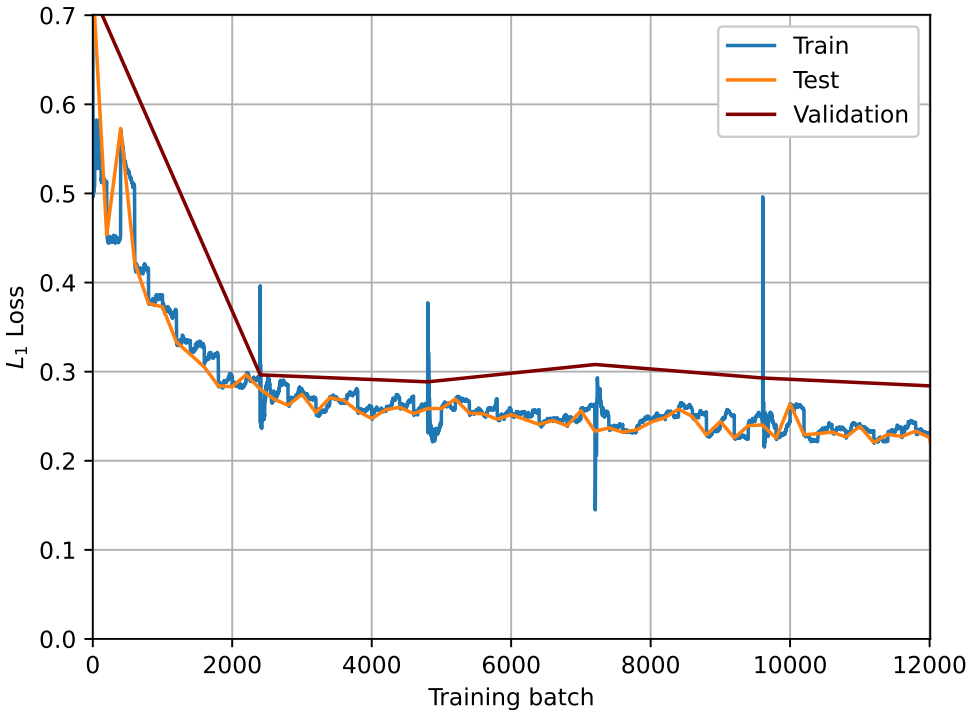

Figure 5.1: Training progression over 5 epochs with batch size 5.

The plot shows similar behaviour to many other neural networks. An initial quick convergence followed by a slow, plateau-like progression most caused by closing into a minimum. The test score remains very close to the training loss suggesting that there is little to no overfitting in the model. The validation score as expected diverges more strongly from the other two since it features completely unseen geometries. The peaks found in the loss curve correspond to the effects of the dropout which at random deactivate 30% of the neurons in the dense layer. In the few spikes that can be seen, it is the case that a set of neurons are blocked that have a large influence on the results. These peaks do not appear in the test or validation score since no dropout is applied during their evaluation.

To draw more conclusions on the model's performance, it is necessary to inspect the results hands-on. For that purpose, several figures have been created that aim to give an unbiased impression of the results. To show the performance for each of the predicted quantities and different angles the first of 5 geometries are taken. The plots, as can be seen in figure 5.2, are composed of two representations of the same geometry. The coloured units found on the faces stand for the prediction values. Their position in the graph indicates the injection location that has been used for the prediction (left) or simulation (right). The colour is used to mark the outcomes for that location and the given gravity vector $\mathbf{g}$. The colour scales can be seen at the bottom of each geometry and are chosen independently for each of the subplots. The reason behind this decision is the often considerable difference in magnitudes of the predictions and simulation results. If the colours were to be measured on the same scale, the profiles would not be visible and they in turn are important when assessing the model's performance. In this case, figure 5.2, violet values correspond to high fill ratios, i.e. simulations where smaller quantities of air remain in the mould and hence suggest a good injection point. Contrary, the yellow dots indicate a smaller fill ratio and hence more air being trapped in the mould. In this case, both the model and the simulations suggest that the upper left facing flange is a bad area to inject the polymer mixture according to the fill ratio.

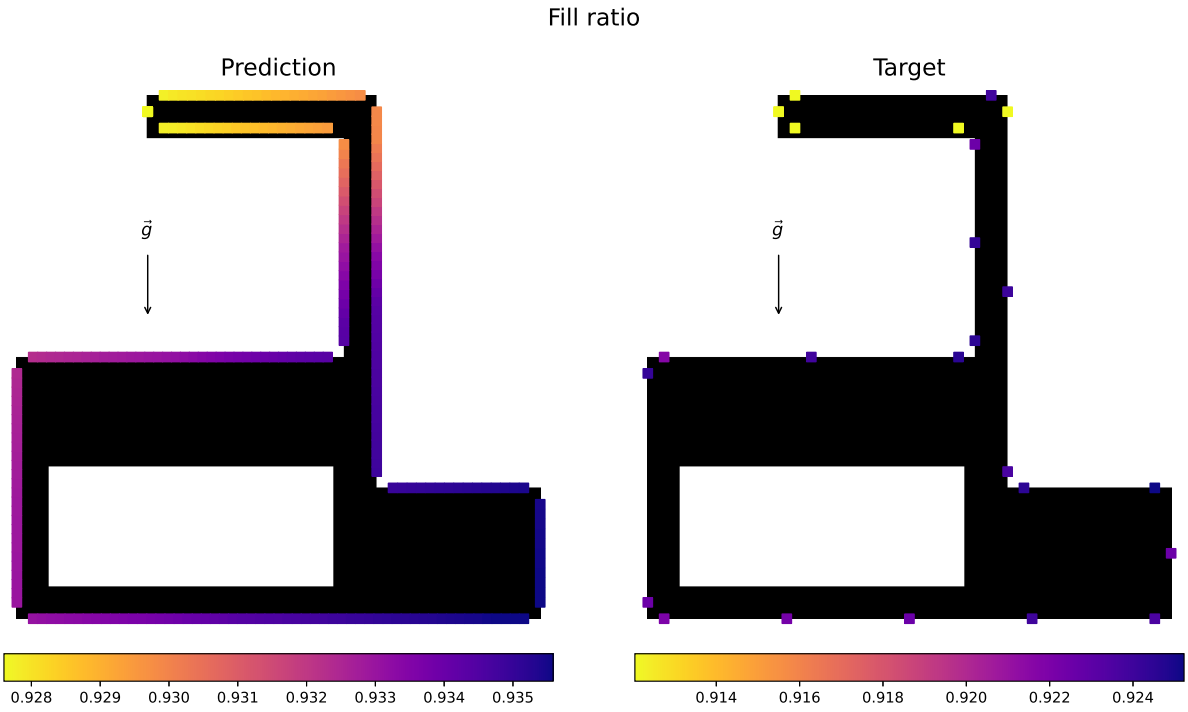

Figure 5.2: Validation case 1: Fill ratio.

The next figure 5.3 instead features the maximal density of the foam. The neural network here predicts a similar profile as for the fill ratio but with an inverse scale. This makes sense from a physical perspective since a small fill ratio also implies that the polymer has gotten stuck in some part of the mould and could not expand. That trapped mass keeps a higher density towards the end of the simulation. The simulations agree with the fact that the worst injection point is likely to be in the upper part of the mould. They however fluctuate unexpectedly in the lower parts of the geometry. It is important to keep in mind that the maximal density is a less reliable unit than for instance the fill ratio. It is calculated as the maximal value over all the cells and is therefore highly susceptible to local numerical errors. The fill ratio on the counter side is an integrated value that benefits from being calculated over the entire volume. This smooths out possible numerical inaccuracies.

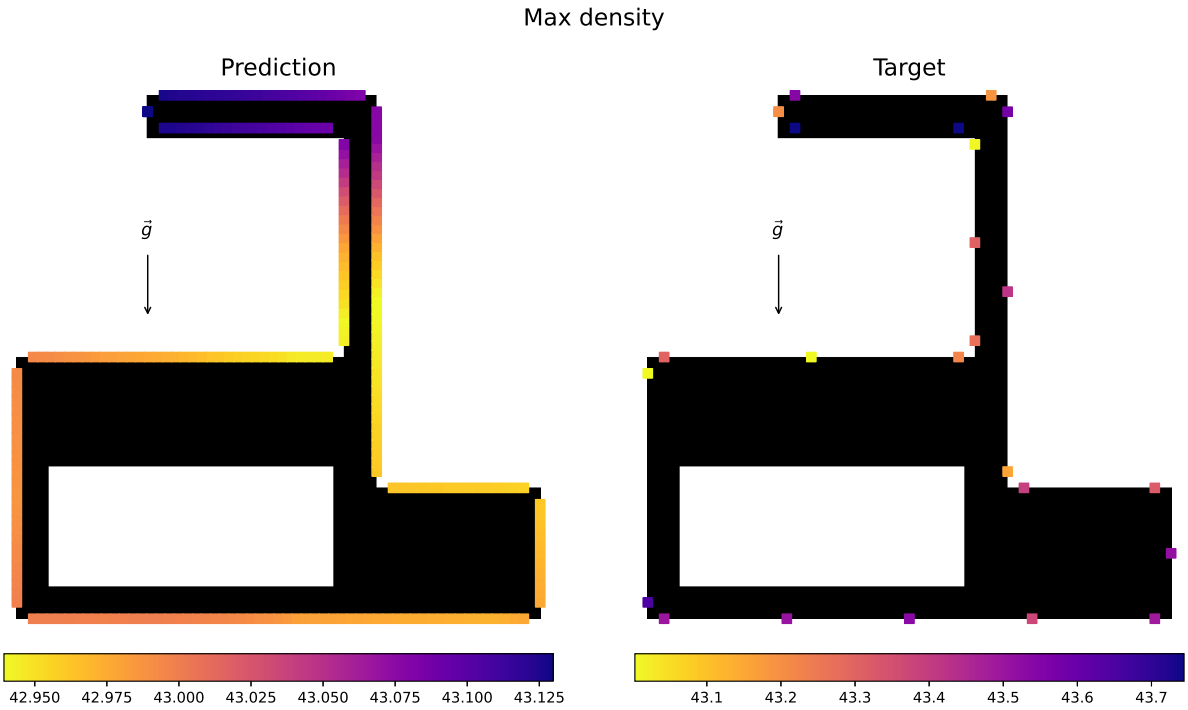

Figure 5.3: Validation case 1: Maximal density.

Next up, the plot showing the results for the density deviation (figure 5.4). From the evaluated quantities, this one is by far the one with the largest errors. The simulation results share the same accuracy issues as the maximal density since the deviation too depends on single-cell values and not on global quantities. It is difficult to find physical explanations for the simulation results. The predicted picture makes little sense since the upper part of the mould should theoretically be a bad injection location and would normally cause larger deviations in the foam density.

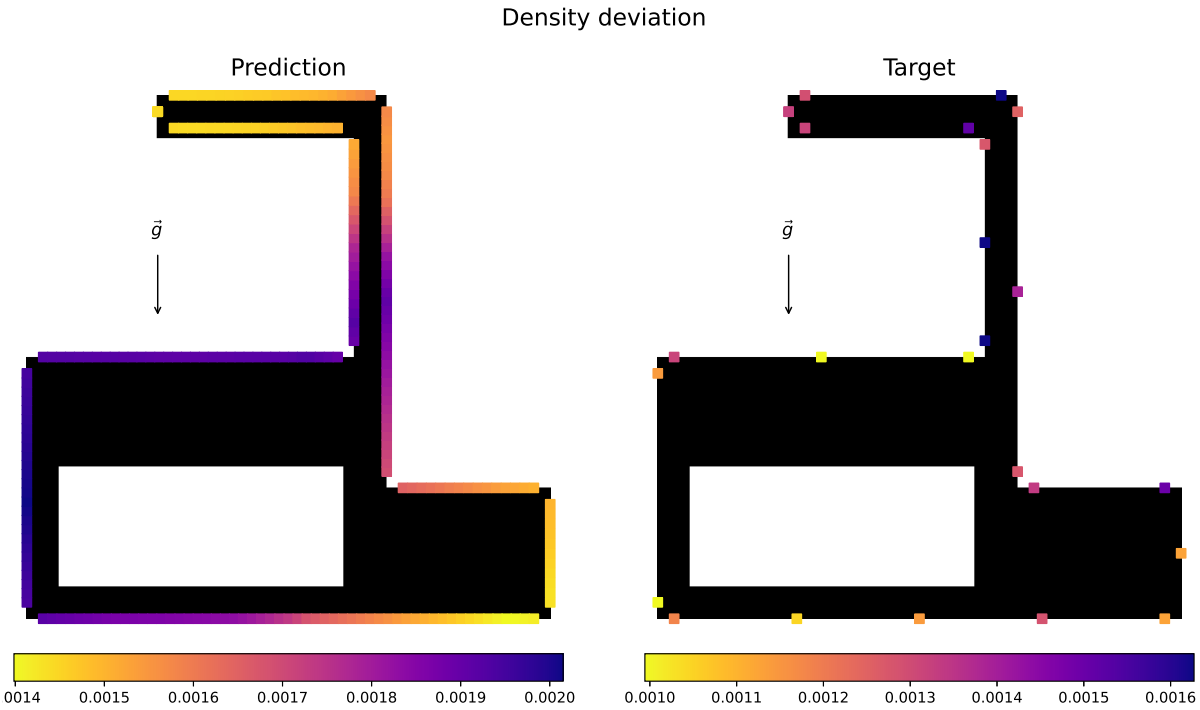

Figure 5.4: Validation case 1: Density deviation.

Getting poor results for both quantities can be a considerable reason to overthink the choice of quality factors. Some other quantities as the density variance could provide a more reliable quality marker. From here on, only the fill ratio will be considered in the results. In figure 5.5 the evaluation angle is set to 90 degrees. As can be seen from the picture, the network shifts its profile, suggesting the lower block to be a good injection region. The simulations agree in the profile, but again miss the magnitude of values by roughly 0.015 at the lower margin. Likely, the model can hardly predict the absolute values but instead tries its best with the value distribution. Although sad, it is important to highlight the actual goal of the optimization, which is to show the best parameter combinations for a given geometry. The exact values are in some sense less important than the overall value profile.

The results have to be judged considering that much fewer points (14 per training geometry) have been evaluated per geometry than the ones seen in the validation cases (25 injection points per geometry). It is therefore important to keep in mind that the network in some sense also misses much of the additional details that can be seen in the validation cases. A positive aspect of the network is that it regardless manages to assemble a credible profile with a rather sparse training background. The smooth profiles, although not always accurate, offer the chance of running simple optimization algorithms that evaluate the net and compute the optimal parameter combination.

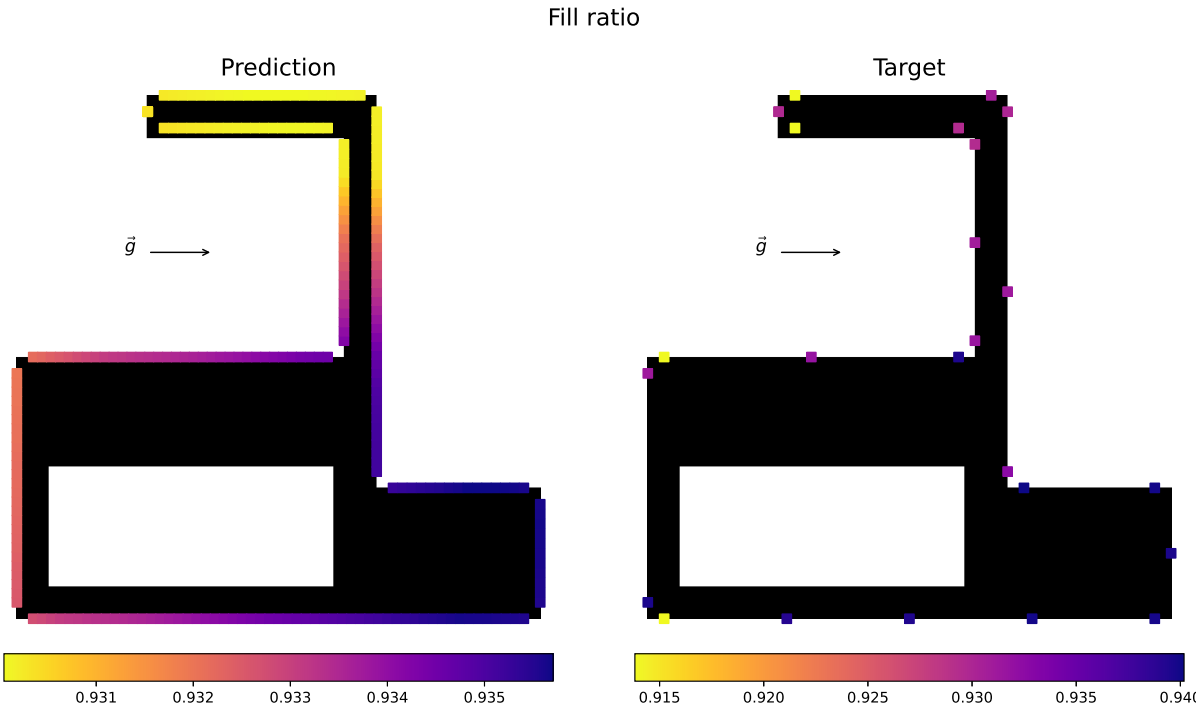

Figure 5.5: Validation case 1: Fill ratio with 90 degree rotation.

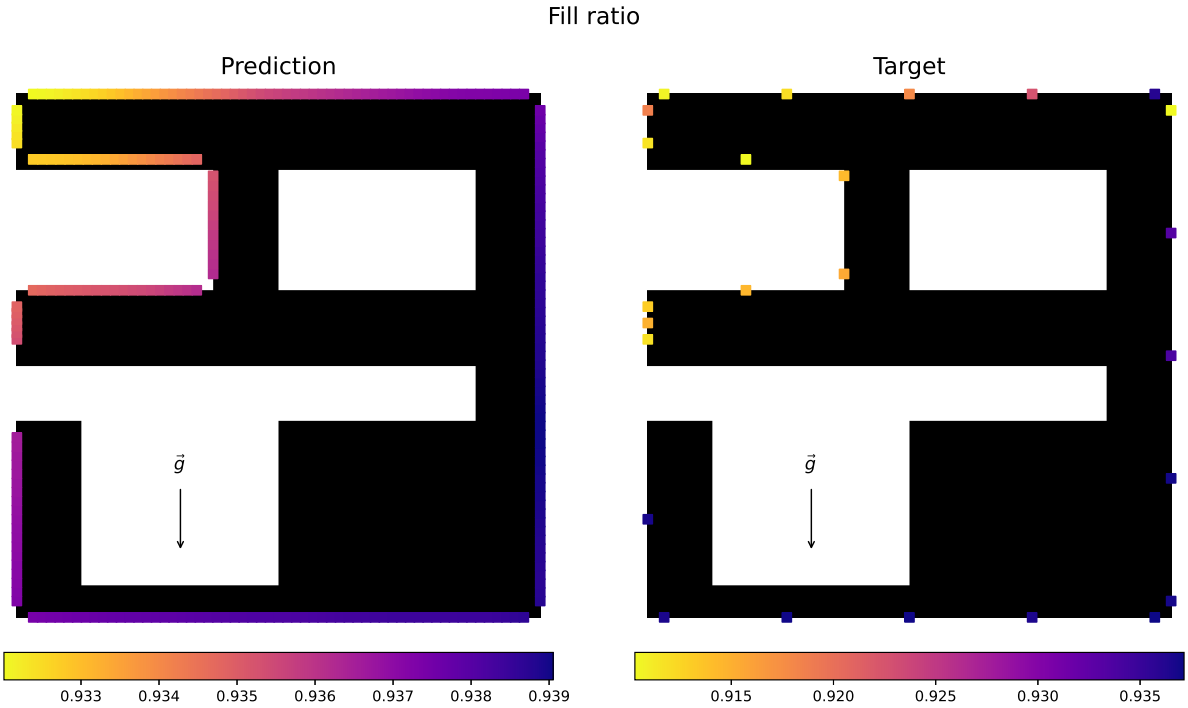

Figure 5.6: Validation case 2: Fill ratio.

In figure 5.6 a different geometry is evaluated for an angle of 0 degrees. The model suggests that the lower right corner is likely to yield a good injection point. Manufacturers of foaming components have a rule of thumb that agrees on both with the prediction and the simulation results: "Inject material at the lowest point with respect to gravity". It is also advisable to inject the polymer mixture in locations that minimize the largest expansion path. This ensures that the foam properties are mostly homogeneous in the whole mould. Generally, the profiles of predictions and simulations do correspond. Again, the model faces considerable discrepancies in the magnitudes.

Another geometry can be inspected in figure 5.6. It resembles a wall structure and could be for instance the insulation of a wall. According to the simulations, a good injection area could be the right wall border. The locations on top of the window's opening on the contrary are expected to lead to bad results. The prediction partially agrees with the simulations under consideration of bad magnitude approximations. It suggests the lower part instead of the right flanch.

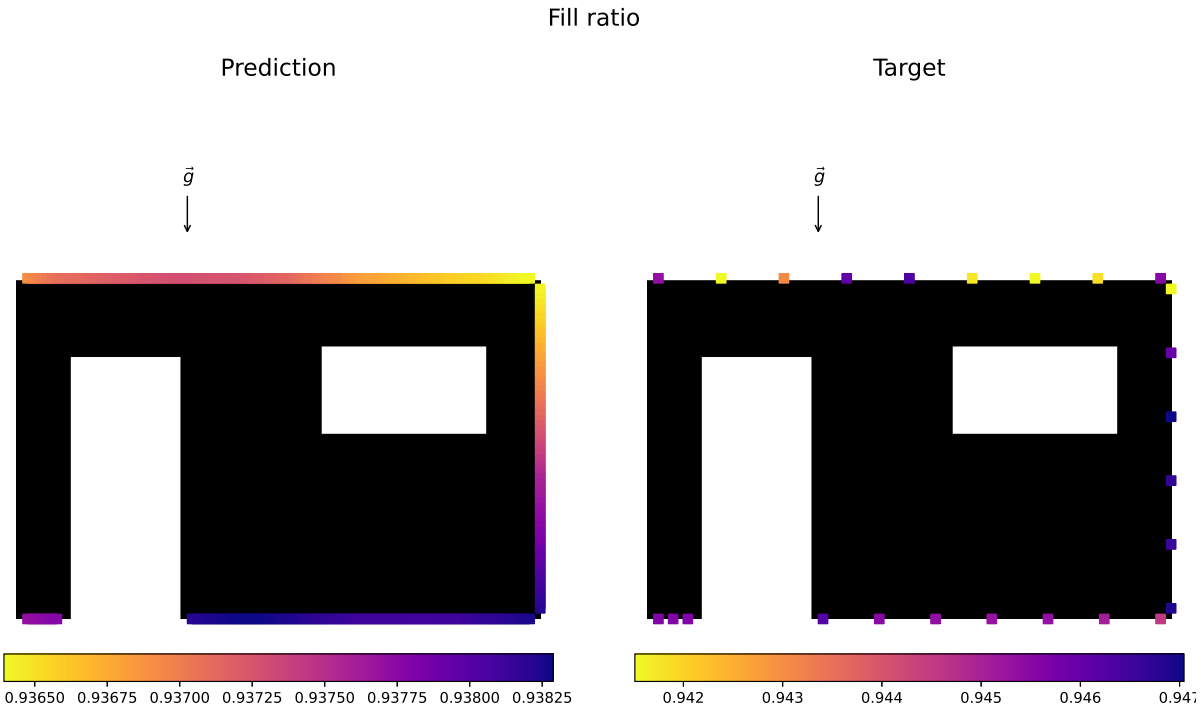

Figure 5.7: Validation case 3: Fill ratio.

Chapter 6

Conclusion

The project was set with the goal of providing a prediction tool that can suggest good simulation settings for foaming operations. Good initial guesses can then reduce the number of high fidelity optimization cycles, thereby shortening development time.

The artificial neural network has been able to process the generated data set to provide reasonable results and prediction maps. The profiles are characterized by smooth transitions and good descriptions of injection locations and angles. Data outliers, such as exceptionally bad simulation results, are rarely described well by the model. This is not regarded as a major inconvenience due to the project goal; finding optimal injection zones rather than bad ones. The model does however in any case substitute simulations or hands-on experiments in its current state. It has been seen that the error in the magnitude is considerable and should not be assumed to be correct by more than one decimal in the case of the fill ratio. The other simulation outputs, the maximal density and density deviation, are more irregular and can often not be explained from a physical point. Consequently, the prediction of these quantities has been irregular and inaccurate. For future implementations, it is recommended to examine and replace these quality responses with other averaged variables that are less prone to numerical errors.

The bottleneck that has impeded the model from being more generally applicable has been the data generation. For a full 3D prediction tool it is necessary to provide more numerous and elaborated simulation settings. The injection locations hereby stand a major difficulty due to the simulation related errors that can arise from ill set locations. The project involves a whole array of tasks, e.g. geometry processing, data sampling and the neural network's architecture, all of which are undoubtedly time-consuming. The subtopics have been dealt with in a brief time and therefore leave room for improvement.

Based on the presented 2D model, an extended 3D tool can be built. The generally low neuron count gives room to extend the convolutional and pooling layers to handle 3-dimensional inputs instead. For a successful training procedure, it is required to provide injection locations that are

distributed along the whole geometry surface. Due to the much larger surface, it is also necessary to sample these points smartly, i.e. maintaining a low number of points while maximizing the coverage. It is also recommended to use additional geometries with possibly less regular shapes than the ones used so far to ensure a good generalization. Other simulation settings, like variable injection rates or different foaming materials, may be also included in these extended versions of the program. Here again, it is important to examine roughly how many additional simulations have to be done for each new feature to obtain satisfying results.

In face of the initial goal and the model's results, the project is seen as successfully concluded.

Bibliography

[1] Aleksandra Kemona and Małgorzata Piotrowska. "Polyurethane Recycling and Disposal: Methods and Prospects". In: *Polymers* 12.8 (2020). ISSN: 2073-4360. URL: https://www.mdpi.com/2073-4360/12/8/1752.

[2] BASF Performance Polymers. *Elastopor: Intelligent sandwich technology for efficient insulation, cooling, storage and transportion*. 2021. URL: https://plastics-rubber.basf.com/global/en/performance_polymers/products/elastopor.html.

[3] Dariusz Niedziela, Ikenna Ireka, and Konrad Steiner. "Computational Analysis of Nonuniform Expansion in Polyurethane Foams". In: *Polymers* 11 (Jan. 2019), p. 100. DOI: 10.3390/polym11010100.

[4] S. Geier, C. Winkler, and M. Piesche. "Numerical Simulation of Mold Filling Processes with Polyurethane Foams". In: *Chemical Engineering & Technology* 32.9 (2009), pp. 1438–1447. DOI: https://doi.org/10.1002/ceat.200900202. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/ceat.200900202. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/ceat.200900202.

[5] S. A. Baser and D. V. Khakhar. "Modeling of the dynamics of R-11 blown polyurethane foam formation". In: *Polymer Engineering & Science* 34.8 (1994), pp. 632–641. DOI: https://doi.org/10.1002/pen.760340804. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/pen.760340804. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/pen.760340804.

[6] Moldex3D R16 material divison. *Viscosity Model for Thermosets (Chemorheology Model)*. 2021. URL: http://support.moldex3d.com/r16/en/standardinjectionmolding_material_materialmodels_viscositymodelforthermosets(chemorheologymode.html.

[7] Rong Zheng Peter K. Kennedy. *Flow Analysis of Injection Molds. Hanser, 2nd Edition*. 2013. URL: https://www.harding.edu/lmurray/themo_files/notes/ch04.pdf.

[8] Wiedemann P. “Wärmeausbreitung in der Mikroskala und numerische Analyse mittels der Methode der diskontinuierlichen finiten Elemente”. In: *Bachelor thesis, TU Berlin* (2012). DOI: https://www.lkm.tu-berlin.de/fileadmin/fg49/AbschlussarbeiteundProjekte/simulation/Bachelorarbeit_PabloWiedemann_Waermeausbreitung_in_der_Mikroskala.pdf. URL: https://www.lkm.tu-berlin.de/fileadmin/fg49/AbschlussarbeiteundProjekte/simulation/Bachelorarbeit_PabloWiedemann_Waermeausbreitung_in_der_Mikroskala.pdf.

[9] Harding University. *Thermodynamic lecture notes*. 2012. URL: https://www.harding.edu/lmurray/themo_files/notes/ch04.pdf.

[10] Alexander Gorban and Iliya Karlin. “Beyond Navier–Stokes equations: Capillarity of ideal gas”. In: *Contemporary Physics* 58 (Feb. 2017), pp. 70–90. DOI: 10.1080/00107514.2016.1256123.

[11] Charles S. Jennifer W. *INTRODUCTION TO PSYCHOLOGY – 1ST CANADIAN EDITION. Chapter 4. Brains, Bodies, and Behaviour*. Vancouver, Canada: BC Campus Open Publishing, 2021. URL: https://opentextbc.ca/introductiontopsychology/chapter/3-1-the-neuron-is-the-building-block-of-the-nervous-system/.

[12] Vandenberghe. *(UCLA): EECS236C - Optimization methods for large scale systems*. 2021. URL: http://www.seas.ucla.edu/%CB%9Cvandenbe/ee236c.html.

[13] Ryan Tibshirani. *Convex Optimization 10-725*. 2012. URL: http://www.stat.cmu.edu/~ryantibs/convexopt/lectures/stochastic-gd.pdf.

[14] Brian Anderson Bullins. “Efficient Higher-Order Optimization for Machine Learning”. In: *Dissertation* (2019). URL: https://www.proquest.com/openview/9aec8bc09671325781386378b2cc819b/1?pq-origsite=gscholar&cbl=18750&diss=y.

[15] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[16] Derya Soydaner. “A Comparison of Optimization Algorithms for Deep Learning”. In: *International Journal of Pattern Recognition and Artificial Intelligence* 34.13 (Apr. 2020), p. 2052013. ISSN: 1793-6381. DOI: 10.1142/s0218001420520138. URL: http://dx.doi.org/10.1142/S0218001420520138.

[17] Jörg Conrad. *Introduction to Artificial Neuronal Networks, Lecture 9*. 2020. URL: http://www.cs.columbia.edu/~mcollins/courses/6998-2012/notes/perc.converge.pdf.

[18] Chigozie Nwankpa et al. “Activation Functions: Comparison of trends in Practice and Research for Deep Learning”. In: *CoRR* abs/1811.03378 (2018). arXiv: 1811.03378. URL: http://arxiv.org/abs/1811.03378.

[19] Xinyu Wang et al. “LBP Based Edge Detection Method for Depth Images with Low Resolutions”. In: *IEEE Photonics Journal* PP (Dec. 2018), pp. 1–1. DOI: 10.1109/JPHOT.2018.2884772.

[20] Keiron O’Shea and Ryan Nash. “An Introduction to Convolutional Neural Networks”. In: *ArXiv e-prints* (Nov. 2015).

[21] Sustainability of Digital Formats: Planning for Library of Congress Collections. *StereoLithography Interface Specification*. 1989. URL: https://www.loc.gov/preservation/digital/formats/fdd/fdd000504.shtml#:~:text=The%5C%20STL%5C%20(STereoLithography)%5C%20file%5C%20format,dimensional%5C%20surface%5C%20in%5C%20triangular%5C%20facets..

[22] Utkarsh Ayachit. *he ParaView Guide: A Parallel Visualization Application, Kitware*. 2015. URL: https://dl.acm.org/doi/book/10.5555/2789330.

[23] Rene K. Mueller. *JSCAD repository*. 2021. URL: https://github.com/jscad/OpenJSCAD.org.

[24] Prost J. *Hands on Hyperparameter Tuning with Keras Tuner*. 2020. URL: https://www.kdnuggets.com/2020/02/hyperparameter-tuning-keras-tuner.html.

[25] Delorme R. “Machine Learning for Efficient Operational Optimization of Energy Systems”. In: *Master thesis, RWTH Aachen* (2020).

[26] Conceptually. *What is Occam’s razor? - Definition and explanation*. 2021. URL: https://conceptually.org/concepts/occams-razor.

Appendix

Additional lemmas and definitions used in the thesis are listed here.

Definition 4 (Expansion Theorem)**.** A continuously differentiable function f may be approximated at a position $\mathbf{w}$ by:

$$f(\mathbf{w}) = \sum_{k=0}^{n} (\mathbf{w}-\mathbf{y})^k \cdot \frac{\nabla^k f(\mathbf{y})}{n!} + \int_{\tau=0}^{1} \dots \int \nabla_{\mathbf{w}-\mathbf{y}}^{n+1} f(\mathbf{y}+\tau(\mathbf{w}-\mathbf{y}))(d\tau)^{n+1} \tag{1}$$

for any $y \in \mathbb{R}^m$. The right hand side is referred to as n^{th}-order approximation.

Lemma 7 (Implication of strong convexity)**.** *A differentiable function f that is strongly convex on the domain $\mathcal{B}$ fulfills:*

$$(\mathbf{w}-\mathbf{y}) \cdot \nabla f(\mathbf{y}) \leq f(\mathbf{w}) - f(\mathbf{y}) - \frac{\xi}{2} \|\mathbf{w}-\mathbf{y}\|_2^2 . \tag{2}$$

Proof. Take (3.55) and divide both sides by t.

$$\frac{f(t\mathbf{w}+(1-t)\mathbf{y})}{t} \leq f(\mathbf{w}) + \frac{1-t}{t} f(\mathbf{y}) - \frac{\xi}{2}(1-t)\|\mathbf{w}-\mathbf{y}\|_2^2 \tag{3}$$

$$\frac{f(\mathbf{y}+t(\mathbf{w}-\mathbf{y}))-f(\mathbf{y})}{t} \leq f(\mathbf{w}) - f(\mathbf{y}) - \frac{\xi}{2}(1-t)\|\mathbf{w}-\mathbf{y}\|_2^2 \tag{4}$$

Taking the limit $t \to 0$ results in the derivative in $\mathbf{w}-\mathbf{y}$ direction:

$$\lim_{t\to 0} \frac{f(\mathbf{y}+t(\mathbf{w}-\mathbf{y}))-f(\mathbf{y})}{t} = (\mathbf{w}-\mathbf{y}) \cdot \nabla f(\mathbf{y}) \tag{5}$$

$$(\mathbf{w}-\mathbf{y}) \cdot \nabla f(\mathbf{y}) \leq f(\mathbf{w}) - f(\mathbf{y}) - \frac{\xi}{2}\|\mathbf{w}-\mathbf{y}\|_2^2 \tag{6}$$

q.e.d

Lemma 8 (Constant boundedness)**.** *For strong convex (ξ-convex), L-smooth function $f : \mathcal{B} \to \mathbb{R}$ it holds that:*

$$0 < \xi \leq L \tag{7}$$

Proof. This relationship results from using the smoothness condition (3.14) and the implications of strong convexity (2):

$$f(\mathbf{w}) \leq f(\mathbf{y}) + (\mathbf{w} - \mathbf{y})^T \nabla f(\mathbf{y}) + \frac{L}{2} \|\mathbf{y} - \mathbf{w}\|_2^2 \tag{8}$$

$$f(\mathbf{w}) \geq f(\mathbf{y}) + (\mathbf{w} - \mathbf{y})^T \nabla f(\mathbf{y}) + \frac{L}{2} \|\mathbf{y} - \mathbf{w}\|_2^2 \tag{9}$$

By merging the inequalities at $f(\mathbf{w})$ it can be seen that the remaining terms boil down to $\xi \leq L$. Since $\xi > 0$ it is also known that $L > 0$. *q.e.d*