universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

„Description of collective magnetization processes with machine learning models“

verfasst von / submitted by

Alexander Kornell, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2021 / Vienna, 2021

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 910
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Computational Science (Master)
Betreut von / Supervisor:	Univ.-Prof. Dr. Norbert Mauser
Mitbetreut von / Co-Supervisor:	Dipl.-Ing. Dr. techn. Lukas Exl

Acknowledgements

I want to thank my supervisors Univ.-Prof Mauser, Dipl.-Ing. Dr. Lukas Exl and Dr. Thomas Schrefl for the guidance during writing this thesis. They provided me great support and without them I would not be able to cover the topics of the work. Furthermore I acknowledge the help of my colleges at the 'Department for Integrated Sensor Systems' who always answered my questions concerning the contents of micromagnetism and machine learning.
In addition I am especially grateful to my parents Sabine Kornell and Johann Kornell as well as to my girlfriend Ella Kormout who greatly supported me during all of my studies. Moreover thanks to my family and friends for always being there when I needed you.

Further, I acknowledge the University of Vienna research platform MMM Mathematics - Magnetism - Materials, as well as the support of the Wolfgang Pauli Institute (WPI) and the project "ROAM" under grant No. P31140-N32.

Abstract

High performance permanent magnets play a key role in many technologies of today's life. They are necessary for cell phones, consumer electronic devices, electrical cars and sustainable energy production. Especially the last two mentioned markets are emerging quickly and therefore the need of permanent magnets will grow significantly in future [46, 52, 67].
The state of the art intermetallic phase of high performance permanent magnets is $Nd_2Fe_{14}B$. Since Neodym is a rare earth element which is mainly mined in China, it is essential to reduce the amount of this element while keeping sufficiently high magnetic properties for their applications [22, 9].

In general high performance magnets are clustered by the magnetic field they can create and the resistance against opposite magnetic fields. These properties are highly related to the macroscopic properties coercive field, remanence and the energy density product which are further depending on intrinsic magnetic properties and the microstructure of the magnet. Looking at these dependence it is clear, that the development of new high performance magnets must be based on microscopic properties. Furthermore it has to bridge the length scales to enable sufficiently exact estimations at the device level. In that sense physical experiments are difficult, or even impossible, to proceed and that is why accurate simulation strategies are necessary.
The continuum theory of micromagnetism operates on a length scale which is small enough to resolve these microscopic features and therefore it is considered to be accurate in predicting the properties of interest. Nevertheless it is worth to mention that classical numerical algorithms based on this theory are either too costly or they need strong approximations to fill the gap between length scales. These are the reasons why modern research in that topic introduces algorithms based on machine learning models which are considered to be very efficient [61, 38, 19].
Nevertheless it is important to understand that accurate machine learning models can only be developed if there is enough training data from experiments or simulations. In that sense, the numerical algorithms which are able to solve the fundamental differential equations of micromagnetism are the basis of the models.
However successfully applying machine learning to calculate the figures of interest of high performance magnets enables to pass microscopic properties through the scales. This leads to a new approach of material design and eventually shows ways to reduce the use of rare earth elements.

This thesis focuses on multigrain microstructures. In general they consist of hundreds to thousands of magnetic grains compressed to a dense body. Numerical calculation schemes based on energy minimization are able to calculate the demagnetization curves efficiently but they are computationally too expensive to accomplish true material optimization. The idea of the machine learning based method described in this thesis is coming from [38]. This work shows how to successfully apply machine learning to proceed time dependent integration of the equation of motion for the magnetization in thin film magnets. However the difference to the work of Dr. Kovacs et al. is that the microstructures used in this thesis are three dimensional and the varying parameter is an externally applied field instead of the time. Therefore it operates in the static environment of micromagnetism which is described by Brown's equation.

It is necessary to note that the scope of this work is to proof the concept of this algorithm for the underlying problem. Therefore it is evaluated on simple microstructures since the simulation of synthetic magnets would exceed the scope of this thesis. However the experiments provide insights to the method and furthermore it is designed in a way that it can be extended to more complex models.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction to High Performance Permanent Magnets

This chapter presents the fundamental physical theory which is applied in this thesis. The first section introduces the key properties of a high performance magnet from a phenomenological point of view and tries to outline their importance in modern applications. Furthermore there is a brief introduction into the micromagnetism which is the foundation to build efficient numerical algorithms to do simulations in that topic.

1.1 Key Properties of High Performance Magnets

The main purpose of a magnet is to create a sufficiently strong magnetic field for an application. In that sense, high performance magnets are needed if the magnetic field for an application has to be either very strong or resistant. This means that high performance magnets can be split into two main categories: the magnetic field they can create and the high resistance against opposing fields.

In general this field is increased with the spontaneous magnetization M_s of the magnetic material and decreases with the temperature T. The first important property of a magnet is the so called Curie temperature T_c which is the temperature where the spontaneous magnetization becomes zero.
For example, the average operating temperature of a permanent magnet in the motor block of a hybrid car is at about $450K$ which is lower than the Curie temperature of $Nd_2Fe_{14}B$ based magnets ($T_c = 558K$) [22]. However the operating temperature is already close the T_c and therefore the magnetic properties are much lower than at room temperature. To mitigate this effect in many application, Neodymium is partially replaced by heavy rare earths elements such as Dysprosium and Terbium [19, 27].

1.1.1 Macroscopic Magnetic Properties

Knowing the temperature of interest, one can characterize a magnet by mapping its magnetization $\mathbf{M}$ to an opposing magnetic field $\mathbf{H}$ or to the magnetic induction $\mathbf{B}$. From a phenomenological point of view the magnetization of the magnet decreases with an increasing opposing field and it is a key feature of a permanent magnet to keep a high magnetization even for a strong opposite field.

Applying this mapping to an ideal magnet and when choosing the direction of $\mathbf{H}$ antiparallel to $\mathbf{M}$, the magnetization stays constant until the coercive field or coercivity H_c is reached. At this point the magnetization flips and for $H := |\mathbf{H}| \geq |H_c|$ it points into the direction of the

opposing field. The demagnetization curve gives the magnetization M which is the projection of $\mathbf{M}$ onto the direction of $\mathbf{H}$ at saturation. The component of the magnetic induction can also be used instead of the magnetization. This is referred as the $B(\mathbf{H})$ curve. Furthermore the value of the magnetization without an opposing field is the so called remanence magnetization M_r or just remanence B_r in the $B(\mathbf{H})$ curve. Last the saturation magnetization M_{sat} is the magnetization of the magnet when it is saturated in a high magnetic field.

For a real magnet the discussed properties can be extracted from the plot of the magnetization depending on the opposing field. In that view the coercivity is the root of M in the second quadrant and further the remanence magnetization is the point where M crosses the ordinate axis. [22, 39]

In summary these are properties of the magnet which have direct influence to the application and therefore it is of main interest to optimize them.

Saturation Magnetization, Coercivity and Remanence

The first macroscopic property to discuss is the saturation magnetization M_{sat}. It is the magnetization of the object when it is fully magnetized. In other words it is the state when an external applied magnetic field cannot increase the magnetization of the material further. If the external field is removed, the magnet is in a metastable state because the magnetization of a ferromagnet tends to be reduced by domain formations [15]. However when the crystallographic preferred direction for the magnetization is different to the field direction, the saturation magnetization is an asymptotic upper bound of the $M(\mathbf{H})$ hysteresis loop in the first quadrant.

The coercivity is a quantity which depends on the intrinsic magnetic properties of the magnet and its microstructure. The maximum value for it is the ideal nucleation field $H_n = 2K_u/(\mu_0 M_s)$ assuming an ideal magnet without defects, local demagnetization fields and thermal fluctuation [70]. In that expression, K_u is the leading term of the magneto-crystalline anisotropy energy which is the work that is required to move the magnetization out of its preferred crystallographic direction [40]. Further μ_0 is the vacuum permeability.

In a real world scenario the mentioned disruptive factors decrease the effective coercivity and only about 30% of the maximum possible value is achieved. In total

$$H_c = \alpha H_N - N_{eff} M_s - H_{th} \tag{1.1}$$

describes the coercive field of a real magnet [15, 22].
Equation (1.1) formulates the macroscopic property in terms of the intrinsic properties and the microstructure of the magnet. In detail $\alpha \in [0, 1]$ is a factor which reduces the ideal coercive field due to defects, misorientations and exchange interactions. Further the constant N_{eff} is related to local demagnetization fields which usually occur at sharp edges and corners in the microstructure. Last the thermal fluctuations are represented in the parameter H_{th} [39, 68].

Furthermore this equation provides the fact that the coercivity is decreasing if the remanence magnetization increases. Remember that the saturation magnetization is the magnetic field where the magnet is saturated which implies that $M_r \leq M_s$ where the equality holds only for the ideal model. In detail this means that the remanence and therefore the strength of the magnetic field of the permanent magnet without an opposing field increases if the saturation magnetization increases. On the other hand the coercive field decreases with the saturation magnetization. In

total this relation provides the fact that a strong resistance against opposing fields comes with the trade-off of a lower magnetic field created by the magnet [15].

Energy Density Product

The unique feature of a permanent magnet is to create a magnetic field by itself, or in other terms, it is able preserve magnetostatic energy in the absence of external fields. Furthermore we introduced high performance permanent magnets in the words of a strong and stable magnetic field and quantified the word stable in terms of the coercivity. Besides that the strength of the field can be classified by the energy density product.
Maxwell's equations propose that the magnetic induction $\mathbf{B}$ is divergence free $\nabla\mathbf{B} = 0$. Furthermore the magnetic field $\mathbf{H}$ is curl free $\nabla \times \mathbf{H} = 0$ without any additional currents in the system. This means that in that setting the volume integral over all space of the product of $\mathbf{B}$ and $\mathbf{H}$ is zero if the corresponding vector fields and scalar potentials are regular at infinity [22]. Before doing the integration, we split the space in the region inside the magnet (1.2) and outside the magnet (1.3) which gives us the following two expressions for B:

$$\mathbf{B} = \mu_0\mathbf{H}, \tag{1.2}$$

$$\mathbf{B} = \mu_0(\mathbf{H} + \mathbf{M}). \tag{1.3}$$

Furthermore we can integrate the product of $\mathbf{B}$ and $\mathbf{H}$ over all space:

$$\int_V \mathbf{B} \cdot \mathbf{H} dV = \mu_0 \int_{V_{out}} \mathbf{H}^2 dV + \int_{V_{in}} \mathbf{B} \cdot \mathbf{H} dV = 0. \tag{1.4}$$

The main point of this equation is that $\mathbf{H}$ an $\mathbf{B}$ must point in opposite directions inside the magnet since the volume integral of $\mathbf{H}^2$ is positive. The terms in equation (1.4) can by compared with the magnetostatic energy outside the magnet given by $E_{mag} = \frac{1}{2}\mu_0 \int_V \mathbf{H}^2 dV$. It follows that the energy associated with the stray field created by the magnet outside of the object can be calculated by:

$$E_{mag,out} = -\frac{1}{2} \int_{V_{in}} \mathbf{B} \cdot \mathbf{H} dV. \tag{1.5}$$

The magnetic induction as well as the magnetic field can be written in terms of a uniform vector field. This simplifies equation (1.5) by replacing the fields by their absolute values $B := |\mathbf{B}|$ and $H := |\mathbf{H}|$.

$$E_{mag,out} = \frac{1}{2} \int_{V_{in}} (BH) dV \tag{1.6}$$

Formula 1.6 provides the facts that the magnetic energy of interest can be increased by either increasing the volume of the magnetic object or by increasing the expression (BH) which is referred as the energy density product of the magnet. This quantity has the units of Jm^{-3}. Moreover it depends on the magnetic induction $\mathbf{B}$ and the opposing magnetic field $\mathbf{H}$ created by the magnet itself. This field is named the demagnetization field $\mathbf{H}_{demag}$ of the magnetic object.

The figure of merit is usually referred as the maximum of the energy density product which is known as $(BH)_{max}$. In general this value is used to characterize magnetic material [15]. It is equal to the biggest square which can be inscribed to the second quadrant of the hysteresis loop of $B(\mathbf{H})$.
However an explicit calculation of the demagnetization field $\mathbf{H}_{demag}$ is hard to archive because it depends on the actual shape of the object. Nevertheless in the special case of ellipsoids,

like spheres, long thin rods and flat plates, the demagnetization field is linearly related to the magnetization of the magnet. For such a shape, the demagnetization field can be written as:

$$\mathbf{H}_{demag} = -\begin{pmatrix} N_x & 0 & 0 \\ 0 & N_y & 0 \\ 0 & 0 & N_z \end{pmatrix} \mathbf{M}, \tag{1.7}$$

where N_x, N_y and N_z are the demagnetization factors along the symmetry axes of the object [4].
In general $N_x + N_y + N_z = 1$ holds. The demagnetization factor parallel to the long axis of a long thin rod approaches zero whereas $N_x = N_y = N_z = 1/3$ in the case of a sphere shaped object [58].
In general a large $(BH)_{max}$ is the goal to achieve and therefore we like to optimize the energy density product according to the demagnetization factor. In the following, we just use N for the demagnetization factor in the direction of the magnetization. By combining the equations (1.3) and (1.7), we receive

$$(BH) = \mu_0|M - NM||-NM| = \mu_0(1-N)NM_s^2, \tag{1.8}$$

which is maximized by $N = 1/2$ assuming that the magnet is saturated without second phases [22, 15].
Furthermore there is a simple approximation for the demagnetization factor $N = 1/(2p+1)$ for magnets in form of a prism which can be used to determine the optimal shape for $(BH)_{max}$ [60]. In detail the prism has the dimension of $l \times l \times pl$ and is magnetized along the axis pl. This leads to the fact that a prism with $p = 0.5$ has a demagnetization of $1/2$ so the energy density product is maximal with respect to the shape, if the object is a prism which is twice as wide as high. This means that the maximum energy density product for an ideal magnet is given by

$$(BH)_{max} = \frac{1}{4}\mu_0 M_s^2. \tag{1.9}$$

In detail this value can only be reached if there is no drop in magnetization for $H > M_s/2$ and when the hysteresis loop of $M(\mathbf{H})$ is squared. In a real world scenario this is not the case, but it is important to keep in mind that an almost squared hysteresis loop and a coercive field greater than the half of the saturation magnetization is the goal to archive when designing high performance magnet with strong magnetic fields [22, 15].

In summary a high energy density product makes it possible to use less magnetic material for applications while keeping the strength of the created magnetic field. This means that the volume and the weight of the needed magnets can be reduced by developing magnets with a large $(BH)_{max}$.

1.1.2 Intrinsic Magnetic Properties

Coercivity, remanence and the energy density product are macroscopic properties of permanent magnets. They are depending on the microstructure of the magnetic object as well as on the so called intrinsic magnetic properties. The most important intrinsic magnetic properties of a magnetic material are the spontaneous magnetization M_s, the Curie temperature T_c, the exchange constant A_{exc} and the magneto-crystalline anisotropy constant K_u.

To recap, the magnetic field of a permanent magnet increases with the spontaneous magnetization. This intrinsic property of the magnet further decreases with the temperature due to thermal fluctuations, whereas the Curie temperature T_c is the temperature where the spontaneous

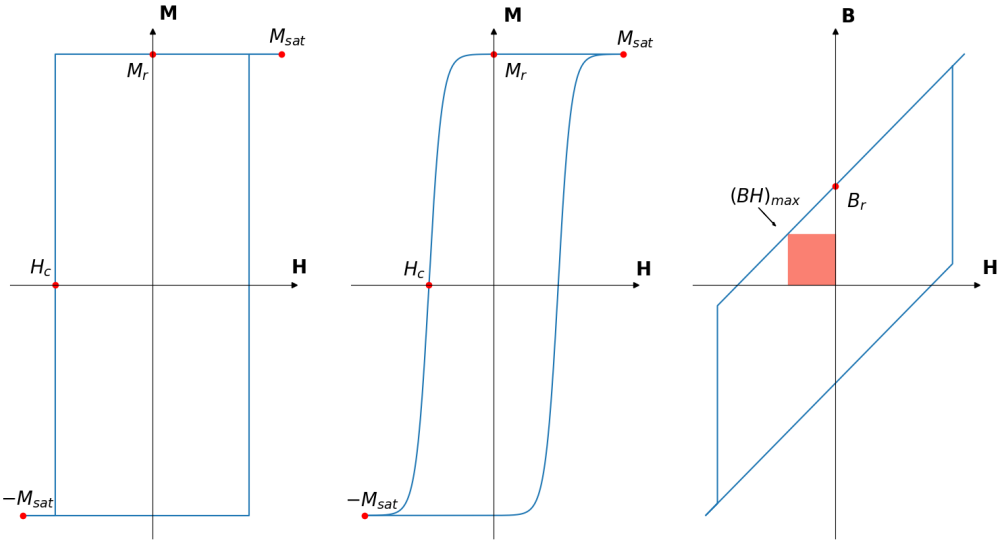

Figure 1.1: Hysteresis loops of (left) ideal magnet, (middle) real magnet, (right) $B(\mathbf{H})$ hysteresis of ideal magnet.

magnetization of the magnet is zero.
The Curie temperature is of main interest for various modern applications. For example a hybrid car has an operating temperature of approximately 450 K in the motor block [22]. This indicates that the used permanent magnets must have a sufficiently high Curie temperature to preserver a strong and stable magnetic field in the motor block.

However the unique feature of a permanent magnet is that it can span a magnetic field in the absence of external magnetic fields. This means that the spins of the atoms which are building up the object are aligned even without an additional magnetic field. This is a key property of ferromagnets in which exchange interactions cause a spontaneous magnetization. Without going into to much detail, it is directly related to the Pauli principal and the Coulomb potential. In general the Pauli principal states that it is forbidden that two electrons of an atom have exactly the same quantum numbers. This further leads to a repulsion acting on two electrons of different atoms but with the same quantum numbers. However in permanent magnets the reduction of the Coulomb potential compensates that effect and therefore it is possible that the object obtains a stable state with aligned spins [3]. In the continuum theory the strength of the exchange coupling is given by the exchange constant A_{exc} which is discussed in section 1.2.3.

The magneto-crystalline anisotropy energy of rare earth magnets is related to their uniaxial crystal structure. Usually they form tetragonal, hexagonal or rhombohedral crystals and the magnetization prefers to lie along the crystallographic symmetry axis. This axis is often called the easy axis of the magnet. The work required to rotate the magnetization away from the easy axis is the magneto-crystalline anisotropy energy. In detail, a deviation of an angle Θ increases the energy of the system by

$$E_{ani} = \int_V K_u sin^2(\Theta) dV, \tag{1.10}$$

where K_u is the leading term of the expansion of the uniaxial anisotropy energy [15]. In this

formula higher order terms of the expansion are dropped because K_u is usually the dominating factor.

1.1.3 Microstructure

The figures of merit, namely the macroscopic magnetic properties, of a magnet are depending on the intrinsic magnetic properties and the microstructure of the object. The question now is how does the microstructure of a high performance magnet look in nature. To answer that question, one has to discuss manufacturing processes of high performance magnets. In general most of the $Nd_2Fe_{14}B$ based high performance magnets which are used in clean energy applications or hybrid electric vehicles are either sintered or hot-deformed $Nd_2Fe_{14}B$ magnets [67]. Both classes have very similar magnetic properties but they differ in their microstructure. In detail, hot-deformed magnets have a finer grain structure than sintered magnets. In this thesis the focus lies on hot-deformed magnets since they are promising candidates to build high performance magnets with less rare earth elements.

However the details of the manufacturing process for hot-deformed magnets can vary since there are on going researches on that topic. One way to start the process is to prepare the raw material by a quenching machine. The porous output is then further be pulverized into a flake powder. This powder consists of a large amount of $Nd_2Fe_{14}B$ crystalline grains with diameters from 10 to 30 nanometers. Furthermore the powder is then pressed at room temperature. The next step is to hot press the cold pressed material which increases the typical grain sizes to 20 to 50 nanometers to get a dense object again. At this point the easy axis of the grains are not aligned and therefore the magnetic properties are not expected to fulfill the necessary conditions of the application. To avoid that problem, the object is then hot-deformed at approximately 800° C. This process usually finishes after a short period of time and at this temperature thermomechanical alignment of the crystal grains occurs. After that step the easy axes are aligned perpendicular to the compressing direction. Moreover the grains are formed to rectangular blocks with a diameter of 200 to 500 nanometers and a thickness of 20 to 50 nanometers.
This manufacturing process enables to craft radially oriented ring magnets as well as axially oriented plate magnets depending on the compression direction of the hot-deformation. Nevertheless it is worth to mention that the crystal orientation mechanism is not fully understand until now [30].

However the main point to take from that section is that the microstructure of hot-deformed high performance magnets consist of grains. Furthermore the geometrical shape of the grains depend on the creation process where the discussed method results in rectangular block structures. Moreover the easy axes of the grains are approximately aligned since this is necessary to provide good macroscopic properties. In addition the sizes of the grains vary from tenths of nanometers to hundreds of nanometers which implies that magnets for applications consist of hundred thousands to millions of such nanocrystalline elements.

1.2 Introduction to Micromagnetism

1.2.1 Fundamentals

So far we discussed magnets from a phenomenological point of view. The question now is how to describe the properties of interest in a theory on which efficient algorithms for simulations can be built on.

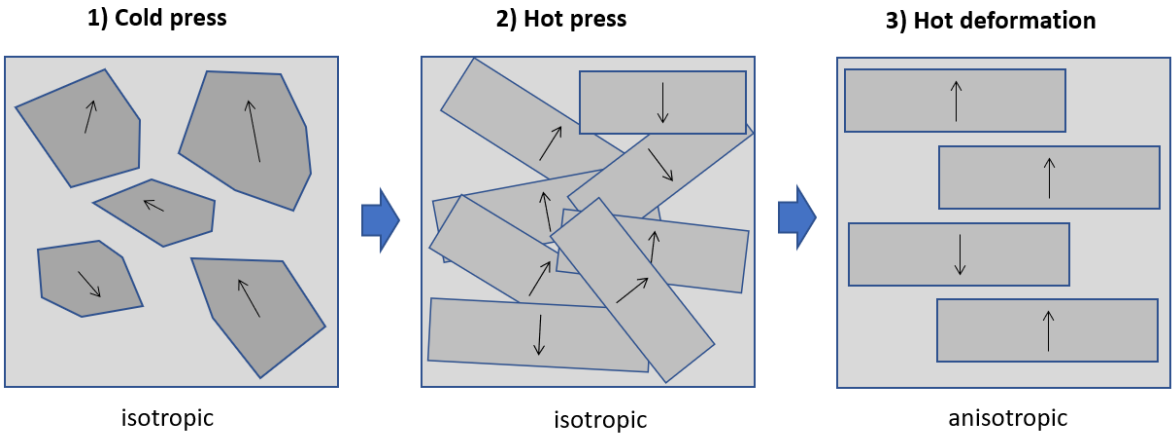

Figure 1.2: Sketch of grains with magnetization during the different stages of manufacturing process; 1) raw material after pulverization 2) rectangular shapes after hot compression 3) aligned grains after hot deformation; Note that the net magnetization is approximately zero after the first two steps since the body is isotropic. Hot deformation is necessary to obtain an anisotropic object with a net magnetization unequal to zero along some axes depending on the material.

In general magnetic phenomenons of ferromagnets are caused by the interactions of spins of elementary particles. This is described by the Quantum Mechanic Heisenberg model [29]. Although this model provides very exact results, it has a lack of scalability in length scales because single spin interactions are taken into account. In detail it would require an enormous amount of particles to represent magnets in the length scale of interest and this exceeds every computational power available [55].

One way to tackle that problem is to describe the state of a ferromagnet by a continuous vector field which is already introduced as the magnetization $\mathbf{M}$. In general this quantity is the fundamental property of interest of the continuum theory of micromagnetism. Furthermore it describes the magnetic moment per volume unit of the object.
To enable a continuum theory of magnetization, one requires the fundamental assumption that the spins of the particles change only by a small angle from one lattice point to the next. In that case the approximation by the continuous vector field is reasonable because one finite element with its associated magnetization can describe a cluster of spins with approximately the same orientation without any large errors.
Furthermore in micromagnetism the absolute value of the magnetization is only dependent on temperature which is further expected to be constant in time and space. This means that the length of the magnetization is uniform over the magnetic object. In addition the direction of the magnetization dependents on time and space. Moreover it is usually written in terms of the spontaneous magnetization as the follows:

$$\mathbf{M}(\mathbf{x}, t) = \mathbf{m}(\mathbf{x}, t) M_s, \tag{1.11}$$

where $\mathbf{m}(\mathbf{x}, t)$ denotes the unit vector [**?**].

The main goal of micromagnetism is to calculate the state of a magnetic body in terms of the magnetization. This quantity is time dependent as already mentioned but there are various use cases where just the equilibrium state is of interest. This is because the characteristic time scale of the dynamic parts of the magnetization can be far smaller than the characteristic time

scales of the changing conditions of the system.
Furthermore it is worth to note that in this work the microstructure as well as the intrinsic magnetic properties of the used materials are expected to be known during the simulations. Moreover the changing parameter is usually the external field since the goal is to predict magnetization reversal processes.

1.2.2 Dynamic Micromagnetism

As already mentioned, the question is how to calculate $\mathbf{M}(\mathbf{x}, t)$. In general the magnetization is a dynamic property which evolves in time and for that case, Landau and Lifshitz provided their fundamental differential equation for a ferromagnet:

$$\frac{d\mathbf{M}}{dt} = -|\gamma|\mathbf{M} \times \mathbf{H}_{eff} - \alpha\frac{|\gamma|}{M_s}\mathbf{M} \times \mathbf{M} \times \mathbf{H}_{eff}, \tag{1.12}$$

where γ is the gyromagnetic ratio and $\mathbf{H}_{eff}$ is the effective field. Equation (1.12) is a partial differential equation which gives the time evolution of the magnetization. This is a space and time dependent vector. In a small particle the magnetization will be uniform due to ferromagnetic exchange. Then the partial differential equation for $\mathbf{M}(\mathbf{x}, t)$ reduces to a system of three coupled ordinary differential equations.
Moreover the first term of the right side of the equation describes the precision of the magnetization around the effective field $\mathbf{H}_{eff}$. In addition the second term is responsible for the relaxation of the system. This means it describes the approach of the magnetization to an equilibrium state. Furthermore the effective field is in general a combination of an external field and the demagnetization field introduced by the magnetic object itself [63].

The Landau Lifshitz equation uses two constants namely the gyromagnetic ratio γ and the damping parameter α. It is possible to estimate a characteristic time scale for the change of magnetization which is related to the Lamor frequency $f = \gamma\mu_0\mathbf{H}/(2\pi)$ for a $Nd_2Fe_{14}B$ based permanent magnet. For such a magnet the term $\mu_0\mathbf{H}$ is typically about a few Tesla and furthermore the gyromagnetic ration is equal to $1.76086 * 10^{11}\text{Ts}^{-1}$. This gives us a Lamor frequency of 28 GHz per Tesla. So the magnetization process happens on a time scale below nanoseconds. This leads to the fact that the system always approaches a metastable equilibrium state if the external magnetic field changes by magnitudes slower than the intrinsic time scale.
Furthermore this means that applications with slowly changing external fields can be accurately simulated by a static approach of micromagnetism due to the fact that the torque on the magnetization approaches zero vastly faster than significant field changes occur.
In detail wind mills rotate approximately 20 times per minute and motors of hybrid vehicles run typically at 1500 (rpm) to 6000 (rpm). This implies that changes of the external magnetic field are happening at a frequency of 1/3 Hz to 100 Hz. This is about 10^{10} times larger than the characteristic frequency of $Nd_2Fe_{14}B$ permanent magnets. This makes simulation techniques based on energy minimization a valid method for these applications [22].

1.2.3 Static Micromagnetism

However to discuss static micromagnetism, it is necessary to do a small excursion to thermodynamics. In that theory the most probable state of a system can be described by the minimum of the Gibbs free energy. Going back to micromagnetism, one can associate the total free energy of a system as the energy which is required to compute the effective field H_{eff}. There are four

major contributions to the total energy of the system [36]:

$$E_{tot} = E_{exc} + E_{demag} + E_{ani} + E_{ext}. \tag{1.13}$$

In this equation E_{exc} is the energy contribution of the misalignment of neighboring spins, usually called the exchange energy. Furthermore E_{ani} is the anisotropy energy. This energy is required to rotated the magnetization out of the preferred crystal axis, namely the easy axis as we have seen in section 1.1.2. Moreover E_{demag} is the energy term produced by the self-demagnetization of the magnetic object. Last E_{ext} describes the the energy related to the external applied magnetic field.
In micromagnetism the key assumption is that the spins change only by small angle from one lattice point to its neighbors. We have used this assumption already in the dynamic case and it is also necessary to take it into account when analyzing the static case. In general this allows to write the energy contributions as volume integrals of energy densities which further gives us more insights to them:

$$E_{tot} = \int_V \left(A_{exc} \sum_{i=x,y,z} \{\nabla m_i\}^2 - K_u(\mathbf{m}\cdot\mathbf{k})^2 - \mu_0 \frac{1}{2}\mathbf{H}_{demag}\cdot\mathbf{M} - \mu_0 \mathbf{M}\cdot\mathbf{H}_{ext} \right) dV \tag{1.14}$$

The first term of this equation describes the exchange energy in terms of the material dependent exchange constant A_{exc}. Furthermore it consists of the sum over all misalignments $\{\nabla m_i\}^2$ of neighboring spin moments. Looking to the sign of this term, one sees that it requires energy to keep a state of the system where are misalignments of spins. In addition the systems energy is also increased if the magnetization of a cell is rotated out of its easy axis $\mathbf{k}$ which is described the second term of the equation. Here we used $\mathbf{m}\cdot\mathbf{k} = cos(\Theta)$, $sin^2(\Theta) = 1 - cos^2(\Theta)$ and we dropped the constant term. Moreover there is the self-demagnetization energy in terms of the self-demagnetization field $\mathbf{H}_{demag}$. This term is highly related to the magnetic reversal process which further is of major importance when discussing the energy density product of a magnet as we have seen in section 1.1.1. The last part of the equation describes the energy contribution of the external field H_{ext} [39].

In the case of static micromagnetism it is expected that the characteristic time scale of the analyzed application is much greater than the intrinsic time scale of the magnetization process. This implies that a significant change of the external field happens only after the magnet reaches a relaxed state. In a relaxed system all torques to the magnetization vanish which can be written as $\frac{\partial E_{tot}}{\partial m} = 0$. The variation of formula (1.14) with respect to $\mathbf{m}$ and under the constraint $|\mathbf{m}| = 1$ gives us the Brown's equation of micromagnetism [11]:

$$0 = \mathbf{m} \times (2A_{exc}\nabla^2\mathbf{m} + 2K_u\mathbf{k}(\mathbf{m}\cdot\mathbf{k}) + \mu_0 M_s \mathbf{H}_{demag} + \mu_0 M_s \mathbf{H}_{ext}) \tag{1.15}$$

Furthermore the effective field can be obtained by the functional derivative of the Gibbs free energy with respect to the magnetization. This is done by dividing the negative second term of the right hand side of equation (1.15) by $\mu_0 M_s$ [36]:

$$\mathbf{H}_{eff} = -\frac{1}{\mu_0 M_s}\frac{\delta E_{tot}}{\delta \mathbf{m}} = \frac{2A_{exc}}{\mu_0 M_s}\nabla^2\mathbf{m} + \frac{2K_u}{\mu_0 M_s}\mathbf{k}(\mathbf{m}\cdot\mathbf{k}) + \mathbf{H}_{demag} + \mathbf{H}_{ext}. \tag{1.16}$$

Equation (1.16) describes the effective field which also appears in the dynamic equation of motion given by (1.12). Indeed the stationary solution of (1.12) implies (1.15). In other words in equilibrium the magnetization is parallel to the effective field and the torque, $\mathbf{M}\times\mathbf{H}_{eff}$ vanishes.

Exchange Energy

The exchange energy is essentially a quantum mechanical effect directly related to the Pauli exclusion principal and the electrostatic Coulomb interaction.
The Pauli exclusion principal states that two fermions cannot have the same quantum numbers. From a phenomenological point of view this means that two electrons with parallel spins are forced away from each other. This force is often called the Pauli repulsion. On the hand this repulsion decreases the Coulomb potential of the system because all electrons have the same charge and therefore the electrostatic potential is lower if they are further apart. In ferromagnets this lowering of the Coulomb potential can be large enough that the preferred state of the system contains parallel aligned spins. This is the reason for a spontaneous magnetization of the object [20].

There are different approaches to describe the exchange energy. One is to assume localized magnetic moments which is done by the Heisenberg model. The exchange energy between two spins i and j is [29]:

$$E_{ij} = -2J_{ij}\hat{S}_i \cdot \hat{S}_j, \tag{1.17}$$

where J_{ij} is the exchange integral and $\hat{S}_i$ and $\hat{S}_j$ are the spin operators. The exchange integral is usually hard to obtain for large systems but for cubic metals it is essentially a constant. Furthermore we are interested in the case of a large number of coupled spins where we can use the classical representation of equation (1.17):

$$E_{ij} = -2JS^2 cos(\phi_{ij}). \tag{1.18}$$

Here S is the spin quantum number of atom i and j.
We further know that the angle between neighboring magnetic spins ϕ_{ij} is small by our fundamental assumption of micromagnetism. This allows us to approximate the equation by a Taylor series of the unit vector of the magnetization around atom i. Using that expansion and the fact that $\mathbf{m}\cdot\mathbf{m} = 1 \implies \mathbf{m}\cdot(\partial^2\mathbf{m})/(\partial x_k^2) = -(\partial\mathbf{m}/\partial x_k)^2$ leads to the exchange energy of a cubic crystal :

$$E_{exc} = \frac{JS^2}{a}\sum_l a^3\left[\left(\frac{\partial\mathbf{m}_l}{\partial x}\right) + \left(\frac{\partial\mathbf{m}_l}{\partial y}\right) + \left(\frac{\partial\mathbf{m}_l}{\partial z}\right)\right], \tag{1.19}$$

where l goes over all unit cells of the magnet with lattice constant a.
Due to the fact that the magnetization is a continuous property, we can further replace the sum of equation (1.19) by an integral over the volume of the sample [20]:

$$E_{exc} = \int_V A_{exc}\sum_{i=x,y,z}\{\nabla m_i\}^2 dV, \tag{1.20}$$

where $A_{exc} \approxeq \frac{JS^2}{a}n_C$. This constant is usually determined experimentally since it is depending on the exchange integral. Furthermore n_C is the number atoms per unit cell which varies with the actual crystal of the magnet [72].

Magneto-Crystalline Anisotropy Energy

We already encountered the magneto-crystalline anisotropy energy in section 1.1.2. The origin of this energy contribution is also from quantum mechanical nature. Without any details, it is related to the coupling of spin moments and the electronic orbital moments and their interactions with the crystal field [39]. This interaction causes preferred axes for the magnetization and it requires work to rotate it out of them. To recap, rare earth magnets usually form tetragonal

or hexagonal crystals. These crystals are uniaxial which further means that they have only one preferred axis referred as the easy axis.
To derive formula (1.10), one needs to express the anisotropy energy density in terms of $sin(\Theta)$ [36]:

$$\epsilon_{ani}(\Theta) = K_0 + K_1 sin^2(\Theta) + K_2 sin^4(\Theta) + ..., \tag{1.21}$$

where Θ is the angle between the easy axis and the magnetization of the particle. It is necessary to note that the odd terms of this sine series are vanishing. Furthermore this formula can be written as follows [72]:

$$\epsilon_{ani}(\mathbf{m}) = K_0 + K_1 - K_1(\mathbf{k} \cdot \mathbf{m})^2 + ... \tag{1.22}$$

since $sin^2(\Theta) = 1 - (\mathbf{k} \cdot \mathbf{m})^2$, where $\mathbf{k}$ is the unit vector pointing into the direction of the easy axis. Moreover the constant terms of this formula are dropped because we are mainly interested in use cases where absolute shifts of the total energy vanish anyway for example energy minimizing. The last thing to do to obtain the anisotropy energy from the anisotropy energy density is to integrate over the volume of the object:

$$\hat{E}_{ani} = \int_V \hat{\epsilon}_{ani} dV \approxeq - \int_V K_u (\mathbf{k} \cdot \mathbf{m})^2 dV, \tag{1.23}$$

where $K_u = K_1$ to be convenient with the previous notation.

Zeeman Energy

The energy contribution of the external field is usually referred as the Zeeman energy. It is caused by the interaction of the magnet with an externally applied magnetic field H_{ext}. This is often the independent variable in simulations based on micromagnetism as we will see at the discussion of the magnetic reversal process in chapter 2.

In micromagnetism each element of the magnet can be understood as a magnetic dipole $\boldsymbol{\mu}$. This means that the energy of the external field is the sum over all products of the dipole moments and the applied field:

$$E_{ext} = -\mu_0 \sum_i \boldsymbol{\mu}_i \cdot \mathbf{H}_{ext}. \tag{1.24}$$

Furthermore this can be transformed to the continuum model by using the magnetization, which is the magnetic moment per volume unit, instead of discrete dipole moments and by replacing the sum with an integral [39]:

$$E_{ext} = -\mu_0 \int_V \mathbf{M} \cdot \mathbf{H}_{ext} dV. \tag{1.25}$$

Demagnetization Energy

The last energy contribution of equation (1.13) is the demagnetization energy E_{demag}. During the discussion of the Zeeman energy, we proposed that the finite elements of the magnet can be interpreted as magnetic dipoles but we dropped the fact that these dipoles also create a magnetic field by themselves. Physics states that the strength of the magnetic field of a dipole decreases with distance and therefore it is a common approach to split it into two parts [3]:

$$\mathbf{H}_{dip} = \mathbf{H}_{near} + \mathbf{H}_{demag}. \tag{1.26}$$

Here $\mathbf{H}_{near}$ is the near field of the dipole approximation. In micromagnetism this field is already covered by the anisotropy constant of the magneto-crystalline anisotropy energy since

the value is usually evaluated experimentally and therefore local dipole moments are taken into account [20]. The other term $\mathbf{H}_{demag}$ is the far field of the magnetic dipole which is referred as the demagnetization field in micromagnetism.

To derive a formula for the demagnetization field it is common to start from Maxwell's equations where eddy currents are neglected. In that case the demagnetization field can be treated statically and the equations of interest reduce to the following:

$$\nabla \times \mathbf{H} = \mathbf{j} \tag{1.27}$$

$$\nabla \cdot \mathbf{B} = 0, \tag{1.28}$$

where j is the current density which satisfies $\nabla \cdot \mathbf{j} = 0$. Furthermore the magnetic field can be expressed by a solenoidal and a nonrational part:

$$\mathbf{H} = \mathbf{H}_{ext} + \mathbf{H}_{demag}, \tag{1.29}$$

with $\nabla \cdot \mathbf{H}_{ext} = 0$ and $\nabla \times \mathbf{H}_{demag} = 0$. Moreover we already defined the magnetic induction by formula (1.2) inside of the object itself. Plugging this relation to Gauss's law (1.28) gives us:

$$\nabla \cdot \mathbf{H}_{demag} = -\nabla \cdot \mathbf{M}. \tag{1.30}$$

In addition $\mathbf{H}_{demag}$ can be written in terms of a scalar potential:

$$\mathbf{H}_{demag} = -\nabla \cdot U, \tag{1.31}$$

since it is free of rotations. Combining (1.30) and (1.31) leads to the following Poisson equation:

$$\nabla^2 \cdot U = \nabla \mathbf{M} = -\rho. \tag{1.32}$$

Finally the scalar potential is defined by:

$$\nabla^2 \cdot U = \begin{cases} \nabla \mathbf{M} & inside \\ 0 & outside \end{cases}, \tag{1.33}$$

because the magnetization outside of the magnetic object is zero as we know from equation (1.3).
However, there is a discontinuity at the surface of the object which needs to be covered by the boundary conditions. In particular the normal magnetic flux into the object must be equal to the normal magnetic flux going outwards the magnet. This condition can be formulated by the following boundary condition [72]:

$$\mathbf{B}_{in} \cdot \mathbf{n} = \mathbf{B}_{out} \cdot \mathbf{n} \implies \mathbf{M} \cdot \mathbf{n} = (\nabla \cdot U_{in} - \nabla \cdot U_{out}) \cdot \mathbf{n}. \tag{1.34}$$

This boundary value problem can now be solved by Greens function which gives us a formula for the scalar potential [20]:

$$U(\mathbf{x}) = \frac{1}{4\pi} \left(\int_{V'} \frac{\rho(\mathbf{x}')}{||\mathbf{x} - \mathbf{x}'||} dV' + \int_{\partial V'} \frac{\sigma(\mathbf{x}')}{||\mathbf{x} - \mathbf{x}'||} dS' \right), \tag{1.35}$$

where $\rho(\mathbf{x}) = -\nabla \cdot \mathbf{M}$ is the magnetic charge density and $\sigma(\mathbf{x}) = \mathbf{M} \cdot \mathbf{n}$ is the magnetic surface.

In addition the energy contribution of the demagnetization field can be calculated analogously to the Zeeman energy as the sum over all energy contributions of each atom:

$$E_{demag} = -\frac{\mu_0}{2} \sum_i \boldsymbol{\mu}_i \cdot \mathbf{H}_{demag}, \tag{1.36}$$

where the external field is replaced by the demagnetization field in formula (1.24). The factor $1/2$ avoids counting pairs of atoms twice. This can be transformed to the continuum form by replacing the discrete magnetic moments by the continuous magnetization and the sum by an integral:

$$E_{demag} = -\frac{\mu_0}{2} \int_V \mathbf{M} \cdot \mathbf{H}_{demag}. \tag{1.37}$$

In summary, equation (1.35) can be used to calculate the scalar potential. Furthermore relation (1.31) gives us the demagnetization field out of that scalar potential. Last, formula (1.37) calculates the energy of the demagnetization field which contributes to the total energy of equation (1.13).

Characteristic Length Scales

So far we discussed the energy contributions to the total Gibbs free energy. However to get a more precise understanding of the terms it is necessary to set them into context and evaluate their relative contributions. In general the relative weights of the right hand side of equation (1.13) change with the size of the magnet.
To get a qualitative understanding, one can start by formulating equation (1.14) in its dimensionless form. Therefore $\tilde{\mathbf{h}}_{ext} = \mathbf{H}_{ext}/M_s$ and $\tilde{\mathbf{h}}_{demag} = \mathbf{H}_{demag}/M_s$ is introduced. This is the dimensionless external field respectively the dimensionless demagnetization field. Furthermore the spatial coordinates are rescaled to $\tilde{\mathbf{x}} = \mathbf{x}/L$ where L is the sample size in one direction. For the sake of simplicity, the volume of the object is assumed as $V = L^3$. Last the total energy is normalized by $\tilde{E}_{tot} = E_{tot}/(\mu_0 M_s^2 V)$. This transforms the energy contributions to their dimensionless forms as the follows [20]:

$$\tilde{E}_{exc} = \int_{\tilde{V}} \frac{l_{exc}^2}{L^2} \sum_{i=x,y,z} \{\nabla m_i\}^2 d\tilde{V}, \tag{1.38}$$

$$\tilde{E}_{ani} = -\int_{\tilde{V}} \frac{K_u}{\mu_0 M_s^2} (\mathbf{k} \cdot \mathbf{m})^2 d\tilde{V}, \tag{1.39}$$

$$\tilde{E}_{ext} = -\int_{\tilde{V}} \mathbf{m} \cdot \tilde{\mathbf{h}}_{ext} d\tilde{V}, \tag{1.40}$$

$$\tilde{E}_{ext} = -\frac{1}{2} \int_{\tilde{V}} \mathbf{m} \cdot \tilde{\mathbf{h}}_{demag} d\tilde{V}, \tag{1.41}$$

where $\tilde{V}$ is the domain of the object after the transformation.

Equation (1.38) introduces the so called exchange length:

$$l_{exc} = \sqrt{\frac{A_{exc}}{\mu_0 M_s^2}}. \tag{1.42}$$

This relation describes the relative importance of the exchange energy with respect to the demagnetization energy respectively the Zeeman energy. In detail the relative contribution of the exchange energy decreases with an increasing sample size as one can see from the factor

$(l_{exc}/L)^2$. This means that small samples tend to show an uniform magnetization due to fact that the total energy increases relatively fast for nonuniform states.

Another important constant is the Bloch wall parameter:

$$l_k = \sqrt{\frac{A_{exc}}{K_u}}. \tag{1.43}$$

This gives a relation between the exchange energy and the magneto-crystalline anisotropy energy. It is essentially the width of the transition zone between two states of magnetization within the magnet. To recap, the exchange energy prefers large distances between two different magnetic domains while the anisotropy energy favors small transition regions [36].

In summary these two parameters can be used to determine an upper bound for a finite element mesh. In detail the spatial discretization must be smaller than both characteristic lengths to be able to resolve magnetic domain walls. Usually these upper bounds limit the mesh size to a few nanometers for hard ferromagnetic materials [72].

Chapter 2

Magnetization Reversal Process

During our discussion of the macroscopic properties of a permanent magnet we introduced the magnetization reversal process from a phenomenological point of view in terms of the hysteresis loop. This loop characterizes magnetic microstructures and make them comparable. In total one can say that the study of the magnetic reversal process is one of the most important techniques to characterize magnetic materials and it is the key to compare results from experiments to simulations. To measure the hysteresis loop of a sample, it is exposed to an external field which aligns the magnetization along the field. Usually the recording of the magnetization reversal process starts with a probe which is fully magnetized along the external field. During the recording the applied field is then reduced until the object is fully magnetized along the reverted axis of the initial external field.
There are already some properties of interest defined in section 1.1. However there are many more characteristics which can be analyzed by the reversal process. For example the magnetization of the multigrain structure can reverse homogeneously or inhomogeneously. Furthermore there can be certain regions within the object where the reversal occurs earlier than in other regions [72]. All these properties are of main interest when building better high performance magnets since a deep understanding makes it possible to improve magnetic microstructures to meet specific requirements of applications.
However, the details of the reversal process usually depend on several different initial conditions and the history of the magnet. For example the hysteresis loop of a fully magnetized sample can look completely different to the loop of a magnet which got its initial state from cooling. Particular importance for permanent magnets is the impact of the strength of the saturation field on the demagnetization curve and the coercive field. Generally coercivity increases with an increased saturation field. In this thesis the main focus lies on the reversal process of initially fully magnetized magnets. Furthermore the loop also varies with the angle between the applied field and the preferred axes within the sample as we have seen during the discussion of the anisotropy energy. Anyhow, it is common to apply the field in z-direction and plot only the z-component of the magnetization as the hysteresis loop.

Plot 2.1 shows the simulated magnetization reversal process of a model magnet. Initially the magnet is fully magnetized along the positive z-axis since there is a strong magnetic field applied along this direction. By decreasing the external field, the curve reaches the remanence magnetization M_r. At this point the magnetization is only caused by the exchange energy, the anisotropy energy and the demagnetization field since there is no external field applied. Afterwards the applied field is increased along the negative z-axis. By increasing the field further some magnetic domains switch until there is no net magnetization anymore [21]. The applied field at this point is the coercive field H_c as we have already discussed in section 1.1.1. Furthermore the magnet can get saturated along the opposite direction if the applied field is even further increased. This saturation is reached at $-M_s$. To complete the full hysteresis loop, the external

field has to be increased again until the magnet is saturated as it was in its initial state.

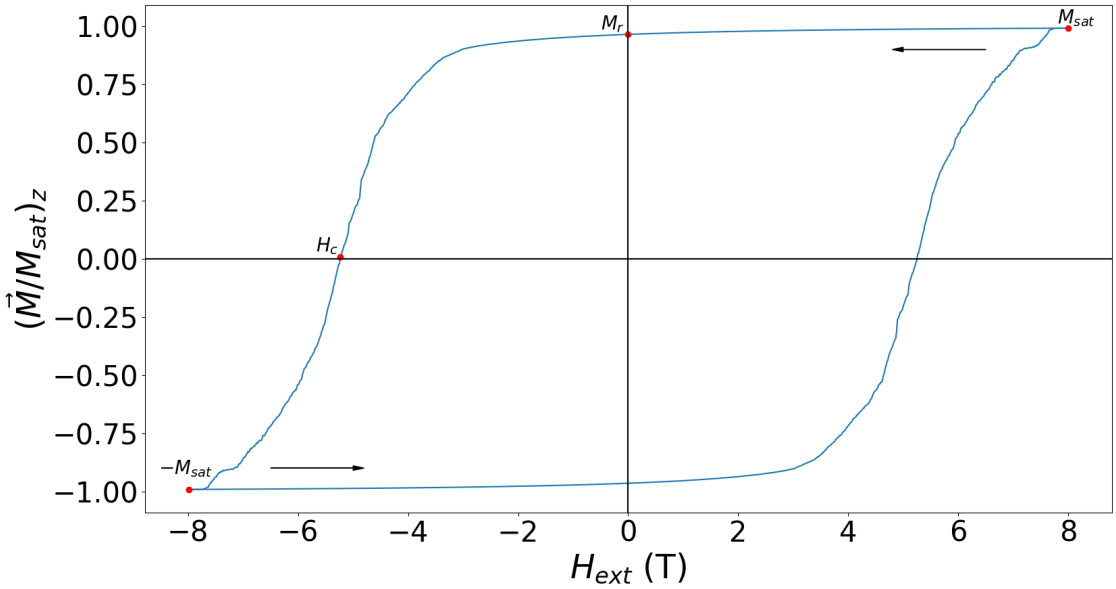

Figure 2.1: Magnetization reversal process of simulated microstructure.

2.0.1 Stoner-Wohlfarth Model

Computing the magnetization reversal process is usually a tough task which needs numerical solvers for the most microstructures. However there is a very simple model introduced by Stoner and Wohlfarth which is analytically solvable. The Stoner-Wohlfarth model describes a magnetic particle which is small enough to only have a single domain and it has an elliptical shape. Although the particle must be very small, it has to be large enough to have a net magnetization. In detail if the size of the particle is too small, thermal fluctuations can switch the magnetization and these changes can cancel each other out although there is only a single domain [72].

To calculate the hysteresis loop of the Stoner-Wohlfarth model analytically, it is necessary to analyze the energy contributions to the Gibbs free energy. First the exchange energy can be neglected since there is only a single domain. Furthermore the demagnetization field of an elliptical particle is uniform. This makes it possible to calculate the magnetostatic energy density by [41]:

$$\epsilon_{demag}^{SW} = \frac{1}{2}\mu_0 M_s^2 N_{\parallel} cos^2(\Theta) + \frac{1}{2}\mu_0 M_s^2 N_{\perp} sin^2(\Theta) \tag{2.1}$$

since the particle is expected to be small enough to have a coherent notation of its magnetization when a sufficiently strong external field is applied. In equation (2.1) Θ is the angle between the easy axis and the magnetization. Furthermore $N_{\parallel}$ and $N_{\perp}$ denote the demagnetization factors of the magnetization parallel and perpendicular to the easy axis. In addition to that, the magnetostatic energy density of the external field is given by [72]:

$$\epsilon_{ext}^{SW} = \mu_0 M_s H_{ext} cos(\Theta_h + \Theta). \tag{2.2}$$

Here Θ_h is the angle between $-\mathbf{k}$ and the applied magnetic field where $\mathbf{k}$ is the direction of the easy axis.

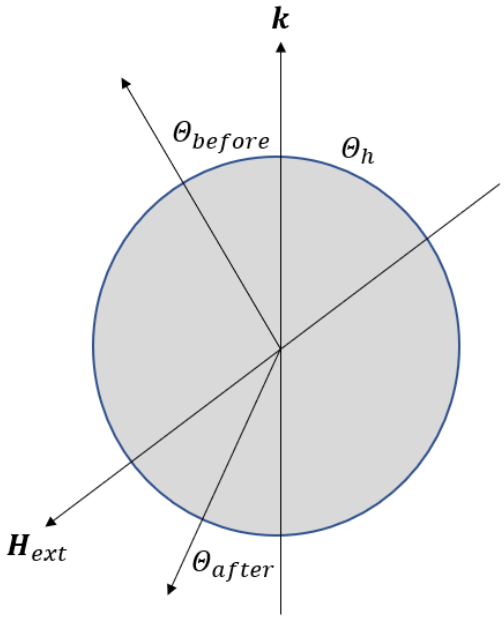

Figure 2.2: Sketch of Stoner-Wohlfarth particle.

Summing up the previously discussed energy contributions leads to the following total energy density of the Stoner-Wohlfarth model:

$$\epsilon_{tot}^{SW} = \underbrace{K_u sin^2(\Theta)}_{\epsilon_{ani}^{SW}} + \underbrace{\left(\frac{1}{2}\mu_0 M_s^2 N_\parallel cos^2(\Theta) + \frac{1}{2}\mu_0 M_s^2 N_\perp sin^2(\Theta)\right)}_{\epsilon_{demag}^{SW}} - \underbrace{\mu_0 M_s H_{ext} cos(\Theta_h + \Theta)}_{\epsilon_{ext}^{SW}}. \tag{2.3}$$

Looking to equation (2.1), one can argue that there is an effective contribution to the anisotropy energy coming from the demagnetization field of the particle. Using this idea one can sum up these two energy contributions by using the effective anisotropy constant coming from the demagnetization field $K_d = \frac{1}{2}\mu_0 M_s^2 (N_\perp - N_\parallel)$ [41]. This leads to a simplified version of equation (2.3):

$$\epsilon_{tot}^{SW} = (K_u + K_d) sin^2(\Theta) - \mu_0 M_s H_{ext} cos(\Theta_h + \Theta). \tag{2.4}$$

The constant terms are neglected since we are interested in the energy minima with respect to Θ.

Figure (2.3) shows a sketch of the energy density landscape in dependence of the angle between the magnetization and the easy axis for different values of H_{ext}. Furthermore it is assumed that $\Theta_h \neq 0$. The plot indicates that there are two stable states for small values of H_{ext}. So for a small external field, the direction of the magnetization is determined by its initial state since there are two local minima. Furthermore the magnetization rotates out of the easy axis when the strength of the external field is increased. This rotation is reversible until the critical field H_n^0 is reached. At this point the particle switches its magnetization irreversible and the first minima of the energy density gets to a saddle point. This field is usually named the switching field of the grain. Moreover the angle between the magnetization and the easy axis enlarges with an increasing applied field until it eventually aligns along the external field direction.

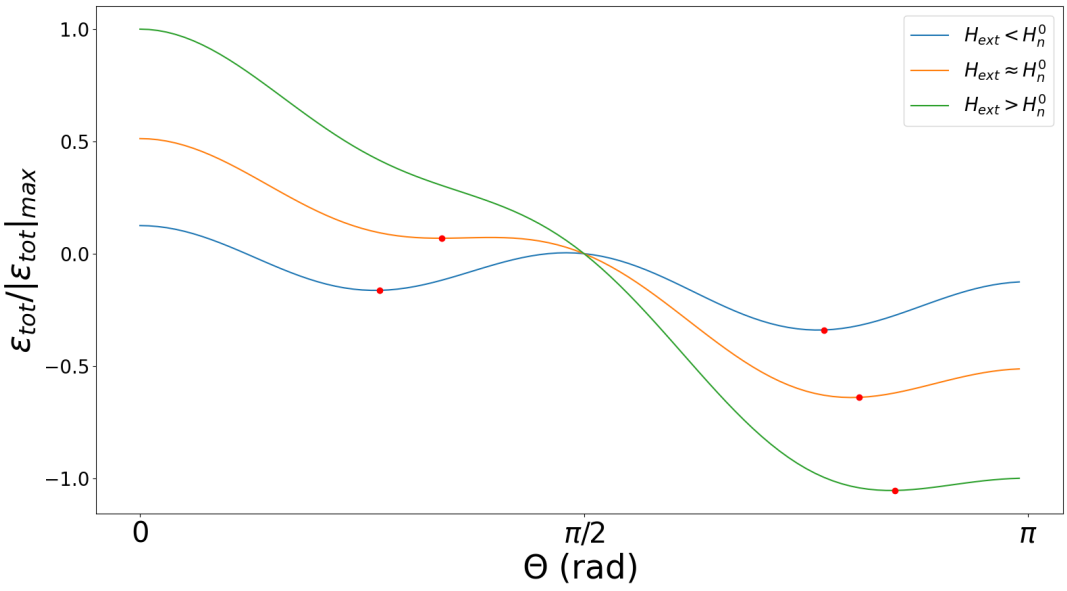

Figure 2.3: Sketch of energy density landscape in dependence of Θ of Stoner-Wohlfarth particle for different values of H_{ext}. Red dots mark equilibrium states [70].

However to analyze the switching field of the Stoner-Wohlfarth particle analytically, it is necessary to start by applying the equilibrium condition to equation (2.4):

$$\frac{\partial \epsilon_{tot}^{SW}}{\partial \Theta} = (K_u + K_d)sin(2\Theta) - \mu_0 M_s H_{ext} sin(\Theta_h + \Theta) = 0. \tag{2.5}$$

Furthermore we already stated that the energy landscape of the Stoner-Wohlfarth particle has a saddle point if the applied field is equal to the switching field. Therefore also the second derivative of the energy density with respect to Θ is set to zero:

$$\frac{\partial^2 \epsilon_{tot}^{SW}}{\partial \Theta^2} = 2(K_u + K_d)cos(2\Theta) - \mu_0 M_s H_{ext} cos(\Theta_h + \Theta) = 0. \tag{2.6}$$

The switching field is then nothing else than the external field which satisfies equation (2.6) for an angle Θ fulfilling equation (2.5).

In detail the external field required to switch the magnetization of the Stoner-Wohlfarth particle is given by [72]:

$$H_n^0 = \frac{2(K_u + K_d)}{\mu_0 M_s}, \tag{2.7}$$

assuming that the external field is exactly anti-parallel to the easy axis of the magnet. This field is also referred as the nucleation field of the magnet which is formally defined as the minimal field which causes the magnetization to rotate out of its preferred axis.
However if the angle between the easy axis and the applied field is $0 < \Theta_h \leq \pi/2$, the field to switch the magnetization of the grain is given by [41]:

$$H_n = H_n^0 (cos^{2/3}(\Theta_h) + sin^{2/3}(\Theta_h))^{-3/2}. \tag{2.8}$$

This formula is used to obtain graph 2.4 which shows the reduced switching field H_n/H_n^0 in dependence of Θ_h. One can see that the necessary field to switch the magnetization is highly dependent on the direction of the external field. In detail the switching field is maximal if the

external field is applied anti-parallel to the easy axis. However it drops to approximately 50% if the external field and the easy axis span an angle of about $\pi/4$.

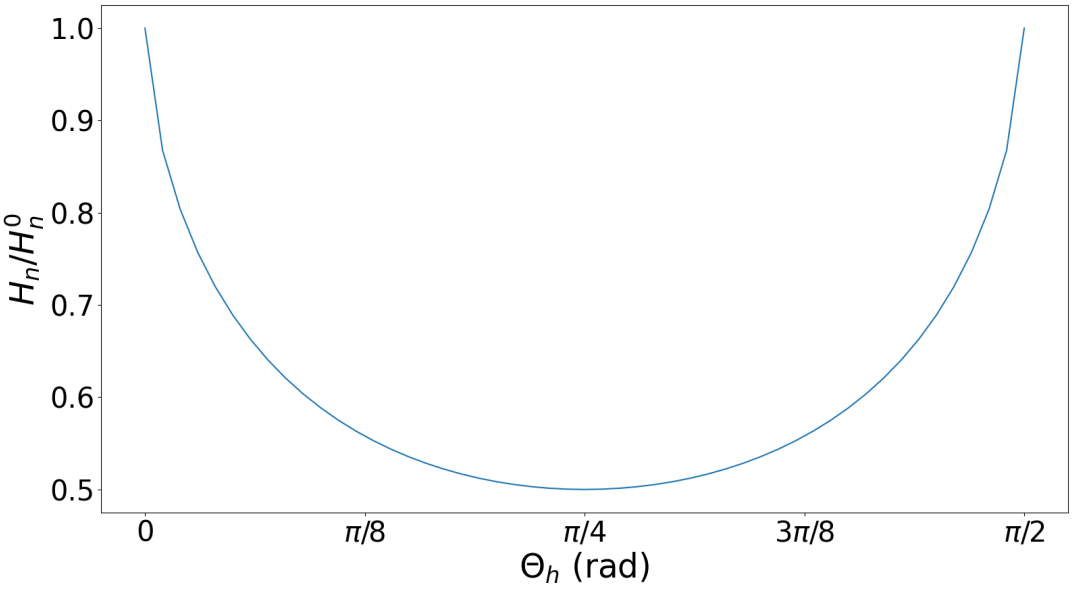

Figure 2.4: Reduced switching field in dependence of Θ_h.

Going back to plot 2.3, one can see that there is a reversible rotation of the magnetization towards the external field before and after the grain switches. This behavior is very important from a practical point of view since it occurs whenever the external field is not exactly anti-parallel to the easy axis of the system. Assuming that Θ_h is small, this rotation can be approximated by the following equations [41]:

$$\Theta_{before} = \frac{\mu_0 M_s H_{ext}}{2(K_u + K_d) - \mu_0 M_s H_{ext}} \Theta_h, \tag{2.9}$$

$$\Theta_{after} = \frac{\mu_0 M_s H_{ext}}{2(K_u + K_d) + \mu_0 M_s H_{ext}} \Theta_h. \tag{2.10}$$

Equation (2.9) describes the rotation for external fields before the grain switches its magnetization whereas formula (2.10) does the same after the magnetization reversal. Before the grain reverses its magnetization, the magnetization rotates away from the easy axis until a critical angle is reached. At this point the magnetization spontaneously changes its direction towards a near direction of the external field. Finally Θ converges to Θ_h if $\mu_0 M_s H_{ext} >> 2(K_u + K_d)$.

The equations (2.9) and (2.10) are small angle approximations. However, it is possible to solve this problem analytically for arbitrary angles. This exact solution can be found in the work [33] of J. Kalezhi and it will be used for the numerical calculations of this thesis.

2.1 Reduced Order Model

So far we mainly focused on theoretical concepts and analytically solvable models. However real world systems are in general too complex to solve them exactly. Therefore it is essential to use numerical solvers and this section outlines a basic algorithm to calculate the magnetization reversal process of more complex structures.

2.1.1 Embedded Stoner-Wohlfarth Model

The first question to answer is on which length scale the model operates. During our discussion of microstructures in section 1.1.3, it is stated that rare earth magnets are usually composed of individual magnetic grains which are separated by a boundary phase. The sizes of these grains are in the range of tens to hundreds of nanometers. Moreover high performance magnets in applications like windmills and hybrid vehicles are generally larger than a millimeter. However this thesis focuses on a length scale of hundreds of nanometers up to a few micrometers.

Models on that length scale consist of multiple grains and therefore multigrain interactions must be counted to calculate accurate estimations. The problem here is that full micromagnetic simulations by finite element algorithms are usually too expensive to compute the magnetization reversal process for a large amount of different structures. The reason for that is that detailed intragrain interactions have to respect the characteristic length scales discussed in section 1.2.3. Nevertheless it is possible to simulate such multigrain structures by algorithms based micromagnetism. In detail the microstructure of interest of this algorithm is built up by Stoner-Wohlfarth particles which are introduced in section 2.0.1. This method is commonly referenced as the embedded Stoner-Wohlfarth model [21]. The main difference to the standard Stoner-Wohlfarth particle is that the field acting on a particle includes the external field as well as the magnetostatic field coming from the other particles. Therefore the angle Θ_h of equation (2.8) is replaced by $\Theta_{h_{tot}}$ which is the angle between $-\mathbf{k}$ and the direction of the total field.

From top to bottom the global microstructure is symmetric and it is built up of cubic grains. This is a rough estimation of real world structures but it can be changed without any further complications. Furthermore the model considers an infinite thin isolating boundary phase which neglects exchange energy contributions between grains. We distribute virtual Stoner-Wohlfarth particles in each grain whereby each particle has the same intrinsic magnetic properties as the grain in which it is embedded. The main thing to mention here is that within a grain, all Stoner-Wohlfarth particles have the same easy axis. Therefore the intrinsic switching field according to equation (2.8) is the same for all particles within the grain. However the field acting on the particle varies since it is a sum of the external field and the local demagnetization fields of the other grains. In addition the switching field of a grain is defined by the reversal of the embedded Stoner-Wohlfarth particles: It is expected that the complete grain reverses its magnetization if the total field at one of the particles is greater than the intrinsic switching field.

2.1.2 Reduced Order Model

In general the idea of the reduced order model comes from energy minimization. Nevertheless the main goal is usually to compute the state of a ferromagnet in terms magnetization. As discussed previously, the model is built up of Stoner-Wohlfarth particles which are analytically solvable. This means that the magnetization can be calculated explicitly and we have approximated this solution already by the equations (2.9) and (2.10). In detail the angle Θ describes the deviation of the direction of magnetization from the direction of the easy axis. Since the easy axis is known when creating the model, it is possible to directly calculate the magnetization of each particle for a given external field. However the external field at an element of the multigrain model is not just the externally applied field since there are interaction by the demagnetization fields for each element of the system.

To avoid any naming complications, the total field $\mathbf{H}_{tot}$ is introduced as the magnetic field at a Stoner-Wohlfarth particle coming from the externally applied field and the demagnetization

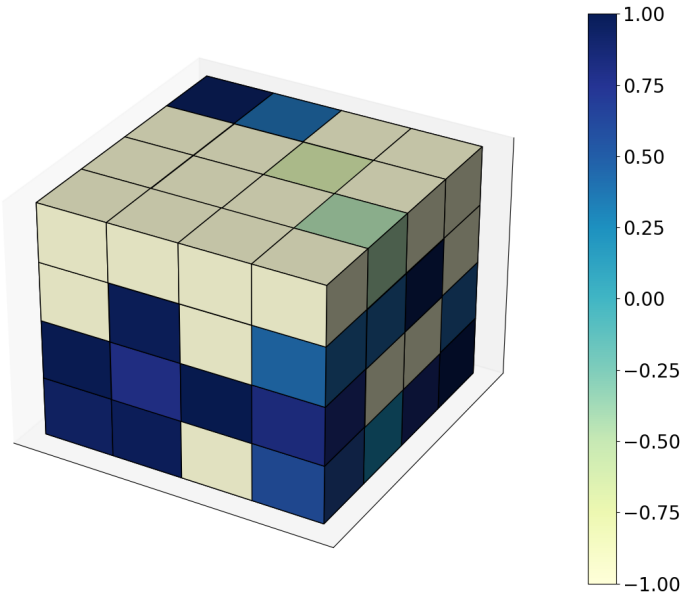

Figure 2.5: Example microstructure of multigrain magnet, color indicates the magnetization along the z-axis.

field of the grains. This total magnetic field can be calculated by [19]:

$$\mathbf{H}_{tot}(\mathbf{x}) = \mathbf{H}_{ext} + \mathbf{H}_{demag}(\mathbf{x}). \tag{2.11}$$

This equation explicitly uses the spatial coordinate $\mathbf{x}$ to underlie that the total field $\mathbf{H}_{tot}$ is eventually different at each discretization point of the system although the external field $\mathbf{H}_{ext}$ is constant within the volume of interest.

Finally it is possible to build up an iterative algorithm to simulate the magnetization reversal process based on these ideas. The first step is to create a multigrain structure where each grain has slightly different intrinsic magnetic properties since this is the case in nature too. In this setup the difference is restricted to the easy axes $\mathbf{k}$ because the model is strictly built of cubes. However these properties of the model are the inputs of the algorithm in addition to an initial state of the magnetization.
Next the size of the external field steps is defined. Furthermore an iteration for each field step is started where the total field is calculated at the beginning. In addition the magnetization is computed by the equations of the Stoner-Wohlfarth model where a check if the total field exceeds the switching field is needed. If this is the case, it is necessary to continue the iteration since the total field for the other grains changes significantly. Last the algorithm can move on to the next external field step if the system reaches a stable state where no more switching fields are exceeded.

Algorithm 1 describes the reduced order model in pseudo code. To clarify uncertainties, it is worth to mention that the strength external field is treated as a scalar value H_{ext} while its direction is given by the unit vector $\mathbf{h}_{ext}$ which is pointing in the opposite direction of the initial saturation field. The index j selects the magnetization or the total field of the j-th particle.

Algorithm 1 Reduced Order Model

Inputs:
$\mathbf{M}_0$, M_s, K_a, K_d, $\mathbf{k}$, $\mathbf{h}_{ext}$

Initialize:
MAXITER, ΔH_{ext}, H_{ext}^{init}, $H_{ext}^{max} > 0$
$H_{ext} \leftarrow H_{ext}^{init}$
$\mathbf{M}[H_{ext}^{init}][0] \leftarrow \mathbf{M}_0$

while $H_{ext} \leq H_{ext}^{max}$ **do**
  for i = 1 to MAXITER **do**
    for SW-particle j in microstructure **do**
      $\mathbf{H}_{demag} \leftarrow$ calculate demagnetization field at SW-particle j
      $\mathbf{H}_{tot} \leftarrow \mathbf{H}_{demag} + H_{ext}\mathbf{h}_{ext}$
      $\Theta_{h_{\mathrm{tot}}} \leftarrow$ angle between total field and $\mathbf{k}$
      $H_{switch} \leftarrow$ equation (2.8)
      if $||\mathbf{H}_{tot}|| > H_{switch}$ **then**
        switch all SW-particles of grain
        $\mathbf{M}[H_{ext}][j] \leftarrow$ equation (2.10) adapted to magnetization
      else
        $\mathbf{M}[H_{ext}][j] \leftarrow$ equation (2.9) adapted to magnetization
      end if
    end for
    if no particle switched **then**
      break
    end if
  end for
  $H_{ext} \leftarrow H_{ext} + \Delta H_{ext}$
end while
return $\mathbf{M}$

2.1.3 Demagnetization Field

A very important computational step of algorithm 1 is the calculation of the demagnetization field in line 6. We already encountered a theoretical discussion of this contribution in section 1.2.3. However a plain implementation of equation (1.31) is in general an expensive task and there are ways to improve the performance.

Within the framework of the embedded Stoner-Wohlfarth method, it is assumed that each grain is uniformly magnetized and that they are separated by an isolating boundary phase. Therefore the integral over the volume or the surface the magnet in equation (1.35) can be split into sums of integrals over the volumes or surfaces of each grain. The volume integrals vanish because $\nabla' \cdot \mathbf{M}(\mathbf{x}') = 0$ within each grain.

$$U^{ROM}(\mathbf{x}) = \frac{1}{4\pi} \sum_{i}^{N_g} \sum_{j}^{N_{s_i}} \int_{\partial V'_{ij}} \frac{\sigma(\mathbf{x}')}{||\mathbf{x} - \mathbf{x}'||} dS'_{ij}, \tag{2.12}$$

where N_g is the total number of grains and N_{s_i} is the number of surfaces of the i-th grain.

Furthermore we can exchange the integration and the nabla operator since $\sigma(\mathbf{x}')/||\mathbf{x} - \mathbf{x}'||$ is continuous on all open sets of ∂V_i and calculate the derivative:

$$\mathbf{H}^{ROM}_{demag}(\mathbf{x}) = \frac{1}{4\pi} \sum_{i}^{N_g} \sum_{j}^{N_{s_i}} \int_{\partial V'_{ij}} \nabla \frac{\sigma(\mathbf{x}')}{||\mathbf{x} - \mathbf{x}'||} dS'_{ij} = \frac{1}{4\pi} \sum_{i}^{N_g} \sum_{j}^{N_{s_i}} \int_{\partial V'_{ij}} \mathbf{M} \cdot \mathbf{n}_{ij} \frac{(\mathbf{x} - \mathbf{x}')}{||\mathbf{x} - \mathbf{x}'||^3} dS'_{ij}. \tag{2.13}$$

Finally it is possible to replace the surface integral by integrals over all edges of the microstructure by again using the fact that the grains are uniformly magnetized. [26] contains the necessary formulas for calculating the demagnetization field for uniformly magnetized polyhedrons.
However this procedure needs some intermediate steps which are out of the scope of this work. Nevertheless it is worth to mention that the necessary equations can be implemented without any further modification but it is computationally better to transform them to an equivalent vectorized formulation. This is especially important when the algorithm is implemented in a high level programming language like Python. Without any further details, the final results are mainly consisting of the following matrix vector product:

$$\vec{H}^{ROM}_{demag} = \underline{\underline{B}} \cdot \vec{M}, \tag{2.14}$$

where $\vec{M}$ and $\vec{H}^{ROM}_{demag}$ hold the magnetization and the demagnetizing field of all grains, respectively. The matrix $\underline{\underline{B}} \in R^{3N_p \times 3N_g}$ contains the spatial components of the structure, where N_p is the total number of Stoner-Wohlfarth particles.
One of the main benefits of this vectorized equation (2.14) of the demagnetization field is that it can be implemented very efficiently on graphic processors since it is possible to highly parallelize the computation [21].

Nonetheless even an optimized algorithm has the drawback that the calculation of the demagnetization field at each iteration has a theoretical runtime of the order $\mathcal{O}(N_g N_p)$ because it needs N_p multiplications for every grain. Going back to physics, one notices that the demagnetization field decreases with increasing distance. This clearly indicates that neighboring grains have an higher impact to the demagnetization field than elements which are far away from each other. This means that grains which are far away from the point of interest can be clustered without a big lack in accuracy.
A similar approximation is successfully applied by the Barns-Hut tree method which is used to

solve N-body problems in astrophysics or electrodynamics [31, 24]. The tree code reduces the theoretical run time to the order $\mathcal{O}(nlog(n))$ and there are libraries available which implement this method. To use this idea, one needs to start by enumerating the grains by their distances and add their values accordingly to $\vec{M}$ and $\underline{\underline{B}}$. This enables to calculate the matrix vector product by hierarchical matrices which are referred as $\mathcal{H}^2$-matrices. The main benefit of the $\mathcal{H}^2$-matrices is that the calculation of the product can even be done with nearly a linear order [13]. One example is the Python package h2tools [47].

Chapter 3

Theoretical Background of Machine Learning

Although the Reduced Order Model introduced in section 2.1 is capable to calculate magnetization reversal processes of multigrain magnets, it still has the problem that simulations on the length scales of applications are very expensive. Furthermore the provided algorithm relies on the embedded Stoner-Wohlfarth model which is a strong approximation to real hot-deformed magnets. In addition it is hard to extend the model with experimental data. One way to tackle these problems is to use machine learning.
The first section of this chapter gives a conceptual introduction to machine learning and tries to clarify commonly used terms.
Furthermore the second section provides an introduction to the Statistical Learning Theory which can be understood as a mathematical framework for common machine learning algorithms. One benefit of this framework is that just a few mathematical concepts are needed to outline the main ideas of machine learning. However I want to note that it lacks when discussing deep learning from a theoretical point of view but this is not the focus of the work.
Finally there is a discussion about the training of machine learning algorithms in practice. Therefore the concept of learning curves and k-fold cross validation is explained.

3.1 Fundamentals

The word machine learning is often used as a synonym for artificial intelligence. However this is not true although both concepts are related. In general these topics are part of computer science and they are the most trending methods for creating intelligent systems. Although there are no formal definitions of these concepts, the main difference is that artificial intelligence can be understood as a concept that tries to develop machines which can mimic human intelligence, whereas machine learning is a method which tries to learn from data without explicit programming. In that sense machine learning can be understood as a subtopic of artificial intelligence. In other words true artificial intelligence has a wide scope and it eventually uses machine learning to solve complex tasks. On the other hand a machine learning algorithm has a very limited task to explore. However this work strictly focuses on machine learning and the main goal is to predict magnetization reversal processes of multigrain magnets.

In general machine learning can be clustered in three classes. Probably the most popular class is supervised learning which is also of main interest of the thesis. Given a set of input and output pairs, supervised learning tries to learn the map from the input to the output by this set of samples. Furthermore the algorithm should then be capable to predict the output of unseen inputs with a sufficiently good accuracy.
A schematic use case of this method is the prediction of toxic material. Initially there is set

of materials which have either the label toxic or non toxic. The machine learning algorithm is trained on this data set. After the training, the method should be able to predict if an unseen material is toxic or non toxic. The word unseen is used in the sense that the new material is not part of the training data. The main benefit of this method is that the prediction of the model is eventually vastly faster than an analytical method to clarify if the material is toxic. However this only makes sense if the accuracy of the machine learning model is high enough compared to e.g. experimental techniques. This further implies that the training set must be able to generalize on the space of interest.
The discussed example is a use case of supervised classification. This means that the output of the map is essentially a class defined by the programmer. On the other hand there is supervised regression. In that case the output variable is continuous and not just a label. To give a practical example, it is assumed that there is a set of data which links properties of trees to their height after 5 years of growth. The goal of the regression algorithm is to learn that relation and then be able to predict the height of a tree in meters. The height of a tree is clearly a continuous variable and therefore supervised regression is the method to choose [49].

Another machine learning method is called unsupervised learning. The essential difference to supervised learning is that there are no outputs within the training data. This means that the algorithm aims to find unknown patterns within the data by itself [25]. There are several use cases of unsupervised learning and some examples are association rule mining, dimensionality reduction and clustering. The first mentioned example is often used in e-commerce to sell customers additional products when they have already added something to their shopping cart.
However in natural science dimensionality reduction methods and clustering algorithms are more important. One example for a dimensionality reduction algorithm is principal component analysis (PCA) [32]. It is assumed that a data set is linearly separable in any dimension, further PCA tries to find a transformation to a lower dimensional space where the transformed data still conserves the information of the original space. In detail this can drastically reduce the space required for the data set by eliminating recurring information. Besides that clustering algorithms can be used to explore outliers in data sets. For example, this can be used to eliminate measurement failures of experimental data. Two examples are the k-means method and the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm.

The third machine learning method is reinforcement learning. The main idea of this method is to reward desired outcomes and punish undesired results of the algorithm. This method is mainly used in gaming and robotics since it still has lacks in applicability in natural science. However to give an example, it is assumed that the algorithm is playing Tic-Tac-Toe. Just to recap, the game starts with an empty 3x3 raster where each of the two player can put either a square or a circle alternately in one empty field of the raster. Initially the algorithm randomly puts its sign to one of the fields whereas a move is rewarded if it leads to a win or punished if it leads to a loss. Furthermore moves which have a higher reward are taken more often from the algorithm than moves with a lower reward. After some time of training, only moves which lead to either a win or a draw are taken from the algorithm since the loosing moves have a very bad score. Although this is just a simple example because Tic-Tac-Toe is only winnable if the opponent makes an error, it should outline the main idea of this machine learning algorithm. However one famous example of an algorithm based reinforcement learning is the chess engine AlphaZero which achieved super-human level within 24 hours of training [62].

However, the used methods in this work are supervised learning and unsupervised learning. Although we already outlined the basic concept of these types of machine learning, it is necessary

to clarify the basic terms used within the topic of machine learning. First, an machine learning algorithm is often called a machine learning model which must be trained before doing accurate predictions. Furthermore one input sample consists usually of multiple features. In general the term feature is used for one component of the input. Going back to the practical example of predicting toxic material, one feature could be the composition of the molecules of the material whereas another one could be the concentration of one specific element. Moreover the output of a machine learning model is usually referred as the prediction. Last, the original data set for training or testing is commonly named the ground truth and this used as a reference to evaluate the accuracy of the model.
In addition to that the term deep learning is mostly used in the context of neural networks. In general a neural network is machine learning model build up of layers where each layer consists of neurons. In that sense, deep learning refers to a machine learning model based on a neural network which has at least three layers. However the transition from classical machine learning to deep learning is indistinct. From a human perspective, the main difference is that classical machine learning approaches like linear regression can be easier interpreted than deep learning approaches. For example if a decision tree classifies a dog on an image, a human can follow the decisions and knows why the model does that prediction. On the hand the reason for a prediction of a multilayer neural network is essentially a black box without any further analysis.

3.2 Statistical Learning Theory

Statistical Learning Theory is a mathematical framework which seeks to calculate predictive functions based on data [64]. To get in touch with this theory, it is necessary start with the basic definition of mathematical learning problem:

Definition 1 ([64])**.** *Let $(\Sigma, G, \mathbb{P})$ be a probability space. Given (Borel measurable) random vectors $X : \Sigma \to \Omega$, $Y : \Sigma \to \mathbb{R}^k$ with $im(X) \subseteq \Omega$ for $\Omega \subset \mathbb{R}^d$ compact.*

For any (Borel measurable) $f : \Omega \to \mathbb{R}^k$ define $\epsilon : \mathbb{R}^k \to \mathbb{R}$, the mathematical learning problem asks for:

$$\hat{f} = \arg\min_{f}(\epsilon(f))$$

This definition means that there is some kind of a data distribution (X, Y) expected. Given that distribution, the learning problem is nothing else than searching for a mapping f which minimizes the function ϵ.

However to do anything with that definition, we need to specify the function ϵ. The first thing to note here is that ϵ is in general called the error of the learning problem and there is no unique mapping for it. Nevertheless one commonly used error function for regression is the least squares error which is defined as the follows:

Definition 2. *For any (Borel measurable) $f : \Omega \to \mathbb{R}^k$, define the least squared error:*

$$\epsilon(f) := \mathbb{E}[(f(X) - Y)^2]$$

In words, the least squared error is the expectation value of the squared difference of the ground truth given by Y and the prediction $f(x)$.

The interesting thing of this setup is that this mathematical learning problem has an analytical solution. In detail $\hat{f}$ is the conditional expectation value of Y given X:

Theorem 1. *Let $\hat{f} := \mathbb{E}[Y|X]$ be the regression function and $\sigma^2 = \epsilon(\hat{f})$. It holds that:*

$$\epsilon(f) := \mathbb{E}[(f(X) - \hat{f}(X))^2] + \sigma^2$$

This further implies that $\hat{f}$ solves the mathematical learning problem given by the definition (1) and the mean squared error.

Proof.

$$\epsilon(f) = \mathbb{E}[(f(X) - Y)^2] =$$
$$= \mathbb{E}[(f(X) - \hat{f}(X) + \hat{f}(X) - Y)^2] =$$
$$= \mathbb{E}[(f(X) - \hat{f}(X))^2] + \underbrace{2\mathbb{E}[(f(X) - \hat{f}(x)) \cdot (\hat{f} - Y)]}_{=0 \text{ because } \mathbb{E}[u(x)Y]=\mathbb{E}[u(x)\hat{f}(x)] \text{ (tower property)}} + \underbrace{\mathbb{E}[(\hat{f}(X) - Y)^2]}_{=\sigma^2} =$$
$$= \mathbb{E}[(f(X) - \hat{f}(X))^2] + \sigma^2$$

□

The question now is if we need machine learning if theorem (1) states that there is an analytical solution for the mathematical learning problem. The answer for that is pretty simple because the exact solution requires us to calculate the conditional expectation value of Y given X. However to do that calculation, it is necessary to know the distribution (X, Y) which we do not know in practice. Nevertheless a machine learning model can be reformulated as a mapping which tries to approximate the theoretical best regression model given by the conditional expectation for the mean squared error [23]. At this point I want to note that theorem (1) uses the mean squared error but it can be extended for other error functions as well [6].

3.2.1 Empirical Risk Minimization

In addition to the fact that it is usually not possible to calculate the theoretical best regressor, it is even not feasible to calculate the true error of any given f because this would also require the calculation of an expectation value where the underlying distribution is again the unknown one. Nevertheless there are ways to approximate that error sufficiently good. To do so, it is necessary to start by defining the empirical mean squared error as the follows:

Definition 3. *Given $z = ((X^{(1)}, Y^{(1)}), ..., (X^{(m)}, Y^{(m)}))$ i.i.d. samples with $(X^{(i)}, Y^{(i)}) \sim (X, Y)$. Define the empirical error as:*

$$\epsilon_z(f) := \frac{1}{m} \sum_{i=1}^{m} (f(X^{(i)}) - Y^{(i)})^2.$$

From this perspective, the main difference of the empirical error compared to the true error is that one can calculate the empirical error by a given set of random samples from the distribution (X, Y). However this set of samples must be drawn in a way that the true error of the underlying distribution is well approximated. This means that the empirical error converges to the true error if the given data set is large enough. Furthermore the samples within the data set have to be independent from each other.
This is actually a very important point to consider when trying to solve a problem by machine learning. There must be a way to draw samples from the underlying distribution. In physics the

preferred way to draw samples are practical experiments. However if the experiments are either very costly or need a long time to proceed, it is very hard to obtain a sufficiently good data set which can build the basis of a machine learning algorithm. To tackle that problem one can extend the data set by simulations but also this technique have to be cheap enough to generate the data set within a acceptable time.

Furthermore there is another problem when solving the fundamental mathematical learning problem. So far it is assumed that we have access to $\hat{f}$ but this is eventually not the case. In general the functions f are named models or hypothesis and they are taken from a functional space called the hypothesis space. This space is formally defined as the follows:

Definition 4. *Let $\mathcal{H}$ be a compact subset of the Banach space $\{f : \Omega \to \mathbb{R}^k, \text{continous}\}$ with norm $||f|| := max_{x\in\Omega}|f(x)|$. We call $\mathcal{H}$ the hypothesis space or the model space.*

The problem is that it is necessary to restrict the hypothesis space to a set of hypothesis which further implies that it is not guaranteed that the best regressor $\hat{f}$ is an element of that space. To give an example, we can define the hypothesis space as the space of all linear regressors but the best regressor is eventually a polynomial of higher order. It is worth to note that the hypothesis space has to have a finite Vapnik–Chervonenkis dimension (VC-dimension) that it is probably approximately correct (PAC) learnable [56].

However the following definition gives us the best hypothesis within $\mathcal{H}$:

Definition 5. *Let $\mathcal{H}$ be a hypothesis space. The best model within this space is defined as:*

$$\hat{f}_{\mathcal{H}} = \arg\min_{f\in\mathcal{H}}(\epsilon(f))$$

In addition to that one can finally define the best empirical regression model as:

Definition 6. *Let $\mathcal{H}$ be a hypothesis space and $z = ((X^{(1)}, Y^{(1)}), \ldots, (X^{(m)}, Y^{(m)}))$ i.i.d. samples with $(X^{(i)}, Y^{(i)}) \sim (X, Y)$, for $i = 1, \ldots, n$. Define the best empirical regression model $\hat{f}_{H,z}$ as:*

$$\hat{f}_{H,z} = \arg\min_{f\in H}(\epsilon_z(f))$$

The main difference between the theoretical best model and the best empirical model is that it is feasible to calculate the best empirical hypothesis without knowing the true data distribution of the problem. However to ensure that the best empirical regressor is a good approximation of the theoretical best regressor, it is required to analyze the difference between $\hat{f}_{H,z}$ and $\hat{f}$.

3.2.2 Bias-Variance Decomposition

One usually starts this analysis by dividing the difference into two parts. This separation is called the bias-variance decomposition, given by the following theorem:

Theorem 2. *Given $\hat{f}_{H,z}$ and $\hat{f}$. It holds that*

$$||\hat{f}_{H,z} - \hat{f}||^2_{L^2} = \Big(\epsilon(\hat{f}_{H,z}) - \epsilon(\hat{f}_H)\Big) + ||\hat{f}_H - \hat{f}||^2_{L^2},$$

where the first term of the right hand side of the equation is called the variance error and the second term is called the bias error.

Proof.

$$||\hat{f}_{\mathcal{H},z} - \hat{f}||_{L^2}^2 = \epsilon(\hat{f}_{\mathcal{H},z}) - \epsilon(\hat{f}) =$$
$$= \epsilon(\hat{f}_{\mathcal{H},z}) - \epsilon(\hat{f}) - \epsilon(\hat{f}_{\mathcal{H}}) + \epsilon(\hat{f}_{\mathcal{H}}) =$$
$$= \Big(\epsilon(\hat{f}_{\mathcal{H},z}) - \epsilon(\hat{f}_{\mathcal{H}})\Big) + ||\hat{f}_{\mathcal{H}} - \hat{f}||_{L^2}^2,$$

The first and the third equalities use theorem (1). □

This theorem gives a very important insight for the practical use of machine learning. It is assumed that the number of training samples is fixed. If the hypothesis space is now enlarged, which essential means that there are more complex models available, the bias error decreases since this term is only dependent on the hypothesis space. However this comes with a trade off since the bias error certainly increases if the size of the hypothesis space is getting larger [16]. A large bias error is referred as underfitting in machine learning. This behavior is seen in practice and it implies that the complex hypothesis space of neural networks essentially requires a large amount of training samples if the networks should outperform classical regressors. In other words this means that neural networks need an efficient way to obtain training samples to be able to build the basis of accurate machine learning models.

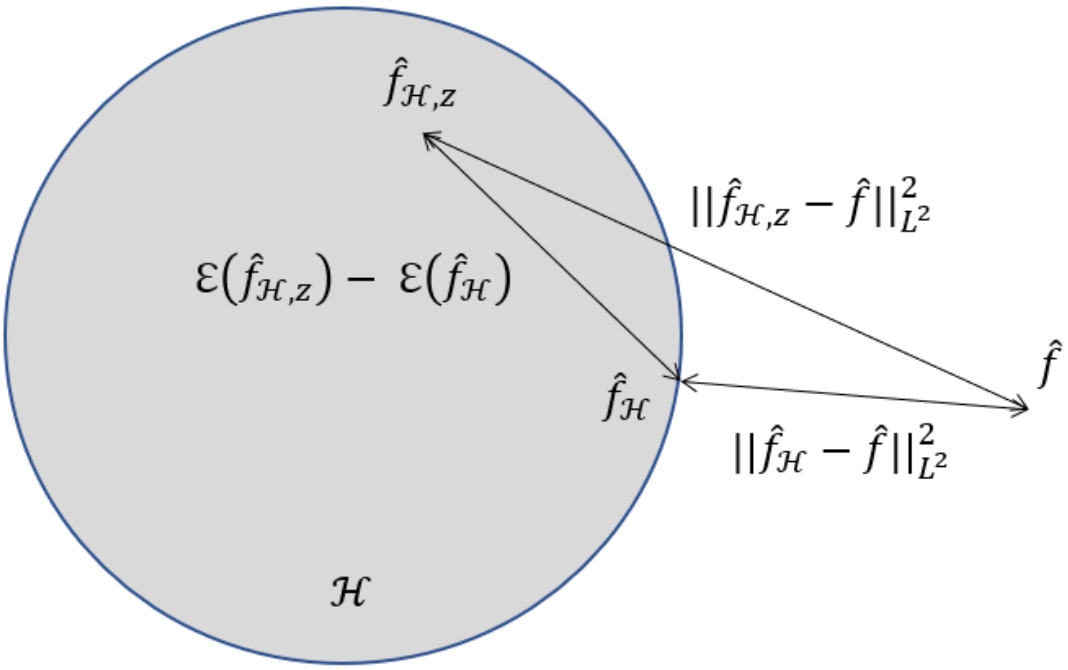

Figure 3.1: Sketch of the bias-variance decomposition in 2 dimensions.

The bias-variance decomposition is introduced as the starting point of the analysis of the difference between the theoretical best model and the best empirical model. The next step of a mathematical analysis of this difference requires to bound the approximation error as well as the generalization error. However a detailed discussion of this topic would exceed the content of this work.
Nevertheless it is worth to note that there are ways to bound the generalization error by mathematical concentration measurements. In that sense it is possible to proof theorems which define theoretical upper bounds for the generalization error in dependence of the number of training

samples. One highly approximated result of such a theorem, proved by the Bernstein inequality, is the following [16]:

$$m \gtrapprox \frac{K}{\epsilon^2}\left[ln(2\mathcal{N}(\mathcal{H},\epsilon)) + ln(\frac{1}{\delta})\right], \tag{3.1}$$

where K is a constant, $\mathcal{N}$ is the covering number of the hypothesis space and δ is a probability. The covering number $\mathcal{N}(S,s)$ is the minimal natural number of discs with radius s which is required to cover the space S.
This relation states that there must be at least m training samples if the generalization error should be smaller than ϵ with probability of $1-\delta$. The essential point to take from that relation is that the number of training samples scales quadratically with the inverse of ϵ. This means that reducing the generalization error by 50 % requires to have four times as many training samples. In addition to that, bounding the approximation error is highly related to the chosen hypothesis space. In general this requires more advanced techniques of mathematics especially when talking about neural networks and that is the reason why this topic is not covered in this work.

3.2.3 Loss Function

So far we discussed machine learning algorithms in terms of the least squared error. However it is possible to prove comparable theorems for different error functions. In machine learning these error functions are known as loss functions. Formally we can define that as the follows [61]:

Definition 7. *Let $\hat{y} = f(x)$ and $\mathcal{H}$ a Hypothesis space. Define an empirical risk minimization problem by:*

$$\hat{f}_{H,z} = \arg\min_{f\in H}\{\mathcal{L}_z(f)\} = \arg\min_{f\in H}\left\{\frac{1}{m}\sum_{i=1}^{m} L(\hat{y}_i, y_i)\right\},$$

where $\mathcal{L}_z$ is the empirical risk and $L(\hat{y},y)$ is the loss function.

In detail a loss function takes the predicted value $\hat{y}$ and the ground truth y and calculates a difference which is important for the underlying problem. It is necessary to understand that there is no 'one fits all' loss function because the importance of the underlying error can vary with the actual problem. However some commonly used examples are the mean squared loss, the cross entropy loss and the hinge loss. Furthermore it is possible to enrich the loss function by information of the underlying problem [38].
One of the main topics of this thesis is to engineer appropriate loss functions to predict the magnetization reversal process. This is done by including parts of micromagnetism, like the preservation of the norm, to this function.

3.2.4 Logistic Regression

The last section introduces the mathematical theory which is needed for the machine learning algorithms used in this paper. However to understand how a machine learning model is trained, it is worth to apply that theory to logistic regression before moving on to neural networks.

For the sake of simplicity a data set where $X \subset R^d$ and $Y \subset [-1,1]$ is considered. This means that each input vector $x \in X$ points to either the label -1 or 1. Furthermore the models are drawn

out of the following hypothesis space: $H_d = \sigma(\langle x, \omega \rangle) : \omega \in R^d$ with $\sigma(z) := 1/(1 + exp(-z))$. A single model of this hypothesis space is fully defined by the vector ω. Given that vector, the model calculates the inner product between ω and the feature vector of the sample. Furthermore σ is applied to that inner product and the output of that function is the prediction. In that case the prediction is a float value within the interval $(-1, 1)$. The actual label of the input sample is then 1 if $\hat{y} > 0.5$ and -1 if $\hat{y} \leq 0.5$.

Training the model can now be understood as the process of finding the ω which best fits the training samples in terms of the loss function. However there are several ways to choose that function. Nevertheless it is necessary to consider the fact that the training set is eventually very large and therefore there must be an efficient evaluation of the loss function to guarantee that training is finished after an appropriate amount of time. That is why one usually uses the logistic loss function based on the Kullback-Leibler divergence [34]. In total this leads to the following empirical risk minimization problem:

$$\hat{\omega} = \min_{\omega \in R^d} \left\{ \frac{1}{m} \sum_{i=1}^{m} L_{KL}(\omega, y_i) \right\} = \min_{\omega \in R^d} \left\{ \frac{1}{m} \sum_{i=1}^{m} log(1 + exp(-y_i \langle x_i, \omega \rangle)) \right\}, \tag{3.2}$$

The goal is now to efficiently solve the right hand side of this equation since the resulting $\hat{\omega}$ defines the best empirical model in our hypothesis space. The first thing to note is that the equation can not be solved analytically and therefore numerical solvers must be considered. Furthermore the number of training samples can be very large which essentially means that direct solver are out of the contest. In total one can say that iterative optimizers are the best performing numerical solvers when training a machine learning model.

3.2.5 Optimizers

We encountered that training a machine learning model requires numerical optimizers. However it is worth to mention that the chosen optimizer can be understood as an hyperparameter of the model. A hyperparameter is essentially a parameter of the model which controls the training. This means that a hyperparameter is not trained but actually defined by the programmer and finding the best hyperparamters is task to consider when solving a problem by machine learning [25].

One very commonly used numerical optimizer for machine learning is the so called stochastic gradient descent algorithm. The main idea of that algorithm is to use the gradient of the loss function to update the weights ω.
This method is essentially a probabilistic version of the standard gradient descent algorithm. The goal of the algorithm is to find a minimum of the differentiable function F. In terms of machine learning this function F is the empirical risk. The following relation describes the fundamental iteration of the gradient descent method [59]:

$$\omega_{t+1} = \omega_t - \eta \nabla F(\omega_t), \tag{3.3}$$

where η is the step size.
The problem of this method is that the evaluation of the full gradient can be an expensive task because there are eventually many training samples to consider especially when using neural networks. However from a probabilistic view one can empirically approximate the expectation value of the gradient by just evaluating the value for some training samples and not for all. This

eventually increases the number of iterations which are necessary to compute the minima but each iteration can be calculated faster. This trade-off usually leads to a lower computation time in machine learning. The following pseudo code outlines the stochastic gradient descent method applied to a machine learning problem:

Algorithm 2 Stochastic Gradient Descent

1: **Inputs:**
   L; X; $\vec{\omega}_0$; k, T, $\eta > 0$
2: **Initialize:**
   $\vec{\omega} \leftarrow \vec{\omega}_0$
3: **for** i = 1 to T **do**
4: X = shuffle(X)
5: $\vec{\omega} = \vec{\omega} - \eta \sum_{i=1}^{k} \nabla_{\vec{\omega}} L(\vec{\omega}, x_i)$
6: **end for**
7: **return** ω

The algorithm takes as an input the loss function L, the training set X, an initial value for $\vec{\omega}$ and the three constants k, T and η. Furthermore it starts the loop, where each iteration first shuffles the data set and then updates $\vec{\omega}$ accordingly to the gradient of the first k training samples. The important point here is that k can be chosen much smaller than the number of training samples and this leads to a significant speedup. After T iterations the last value of $\vec{\omega}$ is returned. However it is also possible to return a mean value of the last few iterations because this eventually improves the result.
Although this algorithm can be very efficiently implemented, it has a major drawback because it is in general not guaranteed that the method finds a global minimum. Nevertheless there are some cases where the convergence to a global minimum of the stochastic gradient descent method can be proofed. For example the algorithm converges to a global minimum if the function to minimize is convex and ρ-Lipschitz. Furthermore recent researches have shown that there are also cases where the stochastic gradient descent algorithm converges to a global minimum within the regime of deep learning although the loss surfaces are usually not convex. One example is that the method is guaranteed to find a global minimum if the used neural network is sufficiently overparametrized [7]. From a practical point of view it is usually a good technique to experiment with different optimizers and different hyperparameters for the algorithms because it eventually happens that a certain setup gets trapped into a local minimum of the loss function.
Although the standard stochastic gradient descent method is already a viable algorithm for training a machine learning model, there are ways to improve the algorithm by using additional information of the function to minimize. One example is the Adam algorithm which uses the momentum of the gradient and an adaptive learning rate to update the property of interest. This method converges faster than the classical stochastic gradient descent method and it has also a higher probability to escape local minima if the loss surface is noisy [35]. However very commonly used machine learning models for dimensionality reduction are convolutional autoencoders based on neural networks. For that type of models there is a further improvement to the Adam algorithm which is named the Nadam method. The Nadam method essentially speeds up the training of a convolutional autoencoder by reducing the number of epochs needed to converge to a minimum [18].

3.3 Training Techniques

It is stated that the best empirical regressor solves the empirical risk minimization on the training set, during the theoretical introduction of machine learning. The problem is that this does not guarantee that the model also performs well on unseen data.

This is for example the case if overfitting occurs. This term means that the model accurately fits the training data but does not generalize well towards different unseen data sets. It often occurs in the practical use of machine learning and this can lead to misinterpreted results. To recap, the goal of machine learning is to predict unseen data accurately and therefore the score on the training set is only conditionally important. However there are different techniques to avoid overfitting in practice.
Besides overfitting it can also happen that the model is underfitting. This essentially means that the hypothesis can not represent the relation between the input and the output data. Underfitting can be prevented by increasing the model complexity since it usually occurs if the hypothesis space is too weak for the underlying distribution. In addition it also occurs if the training data set is too noisy or if the output is related to features which are not represented in the input data. However this must be tackled in the data creation processes.

3.3.1 Learning Curves

The first question is how to identify underfitting and overfitting during the training of the model. Here it is worth to note that these circumstances must be detected at a meta level since it is an intrinsic behavior of a machine learning algorithm. However a very common tool to visualize the learning process of model is the learning curve [25].
There are different ways for plotting a learning curve but it always relates the accuracy of the model to the size of the training set or the number of iterations through the training set. Furthermore it usually contains two graphs where one shows the score on the training set and the other one displays the score on an unseen test set.
In practice one usually starts to develop a machine learning model by splitting the initial data set into two parts. The first part is then used for training while the second one is only taken for testing the model on unseen data. This might seem counter intuitive since one expects that the model should be trained on the largest number of training samples possible but previously unseen data (test set) is required to identify overfitting.

The good point here is that plotting the learning curve is most likely the time consuming part of identifying underfitting and overfitting since its interpretation is straight forward. In detail underfitting is occurring if the accuracy of the model stays constant with an increasing number of independent training samples. In addition there is eventually also a huge difference in the score on the training and the test set. In the case of overfitting the difference of the accuracy on the training and test set increases again after it has approached a minimum at a certain amount of training samples.

Figure 3.2 shows an example learning curve. The graph displays the normalized risk for the test and training set of the model after training it by a certain number of training samples. The risk on both sets reduces until it converges at approximately 7000 training samples. This is the expected behavior during the training of an machine learning algorithm since the model should improve with the number of samples. Furthermore this behavior indicates that the model is not underfitting.
However the minimum risk on the test set is achieved at 8000 training samples. Afterwards the accuarcy on the training set still decreases but the score on the test set increases again. This is a typical indication of overfitting since the algorithm gets biased to the training set.

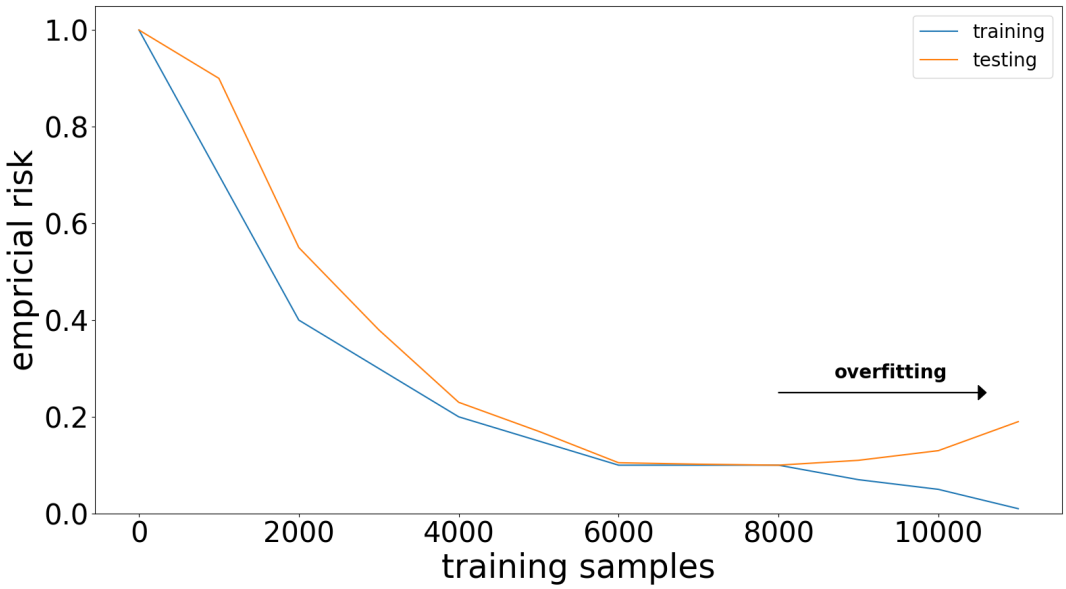

Figure 3.2: Sketch of an example learning curve.

3.3.2 Cross Validation

Although learning curves are already a very good tool to identify overfitting and underfitting, one usually uses another technique in addition to avoid biased models. To motivate this method one just has to take into account that a single learning curve just contains a single test set. This means that it eventually happens that the model is biased to this test set. A method to avoid this behavior is named k-folds cross validation.
Cross validation is in general a technique to estimate the accuracy of the model on unseen data reasonably well by using a statistical analysis on different validation sets [8]. In addition it can be used to find the best hyperparameters for the model. Another benefit of cross validation is that it is easy to implement and most of the standard machine learning frameworks contain an efficient software package for it.

The procedure starts by shuffling the complete training set. The next step is to split the training set in k parts. In this context these k data sets are usually referred as folds. Afterwards one set is taken out and the other sets are used for training. After the training is completed, the model is evaluated on the set which was not used for training. Although this is nothing else than independent test set in this context, it is usually referred as the validation set. The method continues by putting the validation set back into the pool and taking out another fold of the data set. It further resumes by training a new model with the same setup on the new training set and testing it on the different validation set. Finally this iteration continues until every fold is used for validation. A very important benefit of this procedure is that it is easy to interpret from a human perspective. Furthermore the results are usually less biased since outliers can be detected.

The following pseudo code describes an implementation of cross validation. As stated before, the method starts to shuffle the complete training set. Furthermore it iterates though folds by doing the described procedure. Finally it returns all scores for the different validation sets. However there are different versions of cross validation which might exclude the shuffling steps. Nevertheless it is worth to note that without any further information available, one should include shuffling since this in general avoids biases.

Algorithm 3 K-Folds Cross Validation

```
Inputs:
    X; k; Optimizer, Model
Initialize:
    opt ← Optimizer(params) (e.g. SGD, Adam, Nadam)

X = shuffle(X).reshape(k, -1)
for k = 1, ..., len(X) do
    val ← X[k]
    X.pop(val)
    model ← Model(params = hyperparameters)
    model.fit(X, optimizer=opt)
    accuracy ← model.evaluate(val)
    history.append(accuracy)
    X.append(val)
end for
return history
```

3.3.3 Summary

In practice one uses cross validation to identify if a model tends to overfit for certain hyperparameters. By applying this method subsequently for different parameters, one usually can identify a set parameters where the accuracy of the model is reasonably well without overfitting.
However the complete process of training a model should contain cross validation in addition to an independent test set for the final evaluation. In summary a profound way of training a machine learning model should contain the following steps:

- Split the complete data set in training and testing (approximately 20 % should be used for testing)
- Apply k-fold cross validation for identifying the best hyperparameters (k is typically in the range of 5 to 10)
- Training the model with the best hyperparameters on the complete training set
- Evaluating the model on the unseen test data

One can see from this process that a rich data set is the key to apply machine learning successfully. This implies that the data set must contain enough independent training samples. However this means that the data generation process has to be tractable in terms of time and money.

Chapter 4

Artificial Neural Networks

The last chapter introduces empirical risk minimization as a technique to obtain the best model out of a hypothesis space for a given finite data set. Furthermore it describes some numerical optimizers which are the key to successfully train a machine learning algorithm. In addition there is a discussion of the simple hypothesis space of logistic regressors. However the technique of interest of this paper is coming from deep learning based on artificial neural networks. This essentially means that the underlying hypothesis space covers neural networks instead of the logistic regressors.

In this chapter deep learning is introduced in the terms of neural networks. Furthermore there is a description of the commonly used back propagation algorithm which can be used to train a feedforward neural network efficiently. This procedure can be highly parallelized on graphic processors and it is one of the main factors of the success of deep learning in the last decade. Last there is an introduction to a dimensionality reduction method based on autoencoders. This technique is often used in image recognition and it also has potential for physical simulation since high dimensional models like multigrain magnets are often hard to treat in the real space.

4.1 Motivation

The first idea of an artificial neural network was introduced in the 1950s and it was triggered by a work of D.O. Hebb which describes fundamental processes within a biological brain [48]. However modern artificial neural networks diverge from their biological counter parts and there are different types of these networks. In this work the focus lies on artificial feedforward neural networks and for the sake of simplicity, we further refer to this type of model when using the term neural network. Furthermore the used layers of the models are either dense, which implies that they are fully connected, or convolutional.

Nonetheless the relation to biological neural networks is very interesting in terms of artificial intelligence, it is not enough to motivate the use of neural networks within the context of scientific machine learning by this connection. However there is a very fundamental intuition to the use of that hypothesis class also from a scientific perspective. To outline this idea, it is necessary to start with the question if there is an universal best machine learning model. Interestingly enough this can be answered by mathematical theorems namely the 'No Free Lunch Theorems' although the question is rather philosophical [65]:

Theorem 3. *There is no universal best hypothesis space to solve an empirical risk minimization problem.*

Proof. If the underlying unknown data distribution of the empirical risk minimization problem is completely random, it is not possible to infer anything about it from a finite amount of samples. □

However this is a very informal version of the 'No Free Lunch Theorems' but it should outline the main point. It is actually not possible to get insights of a data distribution from samples if it is completely arbitrary.
To finally motivate the use of neural networks we have to rephrase the initial question by: Is it possible to define a hypothesis class which covers most of the other model spaces? In other words this asks for a class where the elements can arbitrarily good approximate the elements of other hypothesis classes. The answer to that question is the model space of neural networks. It is possible to proof that the class of neural networks are 'universal approximators' in the sense that they can approximate classical machine learning models arbitrarily accurate [44, 71]. However a formal theorem for that context requires a fundamental analysis of the architecture of neural networks upfront. Nevertheless it is necessary to note that this does not mean that every other machine learning model is outperformed by a neural network because traditional methods eventually converge faster to the ground truth than their corresponding neural networks. Furthermore they eventually explore properties like linearity of the underlying distribution by their definitions themselves.

4.2 Architecture

The basic element of an artificial neural network is an artificial neuron. This is mathematically defined as follows:

Definition 8. *Given weights $\omega_1, ...\omega_s$, a bias b and an activation function $\sigma : \mathbb{R} \to \mathbb{R}$. Define an artificial neuron by:*

$$f(x_1, ..., x_s) = \sigma\left(\sum_{i=1}^{s} x_i\omega_i + b\right).$$

This means that the output of an artificial neuron is nothing else than the output of the activation function evaluated on a dot product of the weights $\omega_1, ..., \omega_s$ and the inputs $x_1, ..., x_s$ shifted by some bias b. In general the activation function can be defined freely by the programmer. However there are activation functions which make more sense to use than others. In addition the trainable part of the artificial neurons are the weights and biases. Moreover training the artificial neuron is equivalent to searching for the weights and biases which minimize the empirical risk of the problem as it is for logistic regression.
Furthermore it is worth to note that we already encountered the definition of an artificial neuron during the discussion of logistic regression. In general the discussed logistic regression model can be interpreted as a single artificial neuron or an one neuron neural network.

Although single artificial neurons are already machine learning models, one is usually interested in neural networks. An arbitrary neural network consists of a graph which connects artificial neurons [61]. However such an arbitrary neural network is usually hard to model and furthermore it can get very inefficient since the graph eventually contains loops. To avoid such complications, an artificial feedforward neural networks restricts the graph to be directed and acyclic.
However there is a slightly different definition of a fully connected feedforward neural network which will be used in this work [7]. This definition is given by the following:

Definition 9. *Let $L, N_0, ..., N_L \in \mathbb{N}$. A neural network Φ with L layers is the finite sequence of matrix vector tuples*

$$\Phi := ((W^{(1)}, b^{(1)}), ...(W^{(L)}, b^{(L)})) \in \bigtimes_{l=1}^{L} \left(\mathbb{R}^{\mathbb{N}_l \times \mathbb{N}_{l-1}} \times \mathbb{R}^{N_l}\right).$$

$arch(\Phi) := (N_0, N_1, ..., N_L)$ *is the architecture of the neural network* Φ

Here $W^{(l)}$ is the matrix containing the weights of layer l and $b^{(l)}$ is the bias vector of this layer.
The actual machine learning model is then given by the realization of the neural network as:

Definition 10. *Given a neural network* Φ *with* $arch(\Phi) := (N_0, N_1, ..., N_L)$ *and the activation functions* $\sigma_0, ...\sigma_L : \mathbb{R} \to \mathbb{R}$ *continuous where* $\sigma := (\sigma_0, ..., \sigma_L)$*. Define the realization of the neural network as the map* $R_\sigma(\Phi) : \mathbb{R}^{N_0} \to \mathbb{R}^{N_L}$ *continuous with* $R_\sigma(\Phi) := x_L$*.*
x_L *is given by the following scheme:*

$$x^{(0)} := \sigma_0(x),$$
$$x^{(l)} := \sigma_l(W^{(l)}x^{(l-1)} + b^{(l)}) \text{ for } l \in \{1, ..., L-1\},$$
$$x^{(L)} := \sigma_L(W^{(L)}x^{(L-1)} + b^{(L)}).$$

Here $x^{(0)}$ *is the input layer,* $x^{(L)}$ *is the output layer and* $x^{(l)}$ *are the hidden layers of the feedforward neural network.*

This definition does not use the graph explicitly since it defines the layers directly. However it can easily be transformed into code and high level APIs for deep learning like Keras treat neural networks from the layer perspective. In addition one can define the hypothesis space of a specific architecture by $H_{(N_0,...,N_l),\sigma} := \{R_\sigma : arch(\Phi) = (N_0, ..., N_L)\}$,.

4.2.1 Weights

Although the last section contains a formal definition of a neural network, there are some questions left over before going into practice. First the following formula gives the number of weights of a fully connected feedforward neural network Φ with $arch(\Phi) = (N_0, ..., N_L)$ [12]:

$$P(N_0, ..., N_L) = \sum_{l=1}^{L} (N_l N_{l-1} + N_l). \tag{4.1}$$

This formula clearly indicates that a neural network with some hidden layers has many more weights to train than a classical machine learning approach like logistic regression. Assuming the simple example, where the feature vector of the input samples has the dimension 10. In this case the number of weights for logistic regression is also 10 since it is the dimension of the feature vector. However if we assume a simple deep learning model with $arch(\Phi) = (10, 10, 10, 10, 1)$ then the resulting number of weights and biases for the neural network is 341 which is by far larger than the comparable logistic regression model. Intuitively one can see that neural networks can realize more degrees of freedom of the underlying data distribution but it also requires more training samples to adjust the parameters accurately.

4.2.2 Activation Functions

Another very important component of a neural network are the activation functions. The definition assumes the same activation function for all artificial neurons within a layer which is usually done in practice too. However the question is which functions are useful for the tasks to solve. The first thing to note here is that all calculations within a neural network are linear besides the evaluation of the activation function. So if the model should accurately predict nonlinear relations, the activation function of at least one layer has to be nonlinear. Furthermore the function is eventually evaluated very often which further implies that it must be possible to do this sufficiently efficient. Last it should be possible to calculate the gradient of the activation function without any further complexity since a neural network is usually trained by a numerical optimizer based on gradients. However there are various functions which satisfy these restrictions and there is again no 'one fits all' solution for that problem. In detail the activation functions can also be interpreted as hyperparameters of the neural network.
However there are some examples of commonly used activation functions [53]:

The logistic regression or sigmoid function is usually used in the output layer of neural networks used for classification. The evaluation can be interpreted as probability for a sample to have a specific label. We already encountered that function during the discussion of the logistic regression class where we applied machine learning to a binary classification problem. The sigmoid function is given by the following formula:

$$\sigma_s(x) = \frac{1}{1+e^{-x}}, \tag{4.2}$$

$$\frac{\partial \sigma_s(x)}{\partial x} = \sigma_s(x)(1-\sigma_s(x)). \tag{4.3}$$

Another commonly used activation function is the rectified linear unit (ReLU). It is usually the default activation function for the hidden layers of a deep learning model based on neural networks. The main benefit is that layers with that activation function are easy to optimize since their evaluation do not require the calculations of exponential functions or divisions [2]. It is given by the following formula:

$$\sigma_{ReLU} = \begin{cases} 0 & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}, \tag{4.4}$$

$$\frac{\partial \sigma_{ReLU}(x)}{\partial x} = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ undefined & \text{if } x = 0 \end{cases}. \tag{4.5}$$

The last example to discuss is the exponential linear unit (ELU) which is a variant of the ReLU activation function. Experiments on the MNIST data set show that the ELU activation function speeds up the training. Furthermore it increases the accuracy of the model compared to the standard ReLU activation function. In addition it alleviate the vanishing gradient problem which eventually occurs when using the ReLU function [14]. This function is given by the following formula:

$$\sigma_{ELU} = \begin{cases} \alpha(e^x - 1) & \text{if } x \leq 0 \\ x & \text{if } x > 0 \end{cases}, \tag{4.6}$$

$$\frac{\partial \sigma_{ELU}(x)}{\partial x} = \begin{cases} \alpha e^x & \text{if } x < 0 \\ 1 & \text{if } x > 0 \\ 1 & \text{if } x = 0 \text{ and } \alpha = 1 \end{cases}, \tag{4.7}$$

However these are just three examples of activation functions. There a various different methods out in the world which may fit better to the actual problem to solve [53].

4.3 Universal Approximation Theorem

It is stated that one neuron feedforward neural networks are equivalent to logistic regression models if the sigmoid function is used as the activation function. However during the motivation of neural networks, it is explained that this is possible for other hypothesis spaces too. This behavior is formally described by the universal approximation theorem [5]:

Theorem 4. *Let $\Omega \subset \mathbb{R}^{N_0}$ compact, $U : \Omega \to \mathbb{R}^{N_L}$ continuous, $\sigma_0 : \mathbb{R} \to \mathbb{R}$ not polynomial, $L \in \mathbb{N}$ with $L > 1$ and $\epsilon > 0$. Then there is a neural network $\Phi \in H^{\sigma_0}_{(N_0,\dots,N_L)}$ with*

$$\sup_{x\in\Omega} |U(x) - R_{\sigma_0}(\Phi)(x)| \leq \epsilon.$$

The proof of this theorem can be found in [44] since it would exceed the content of this work. Before going into the discussion of this theorem, it is worth to note that Prof. Ding-Xuan Zhou extended this theorem for convolutional neural networks [71]. This is of main interest for that work because the algorithm provided in chapter 5 relies on convolutional autoencoders.

However theorem 4 states that a feedforward neural network with at least one hidden layer and a non polynomial activation function is enough to approximate every smooth function within the space of interest. In other words we can approximate every other hypothesis class by the hypothesis class of neural networks with at least one hidden layer. Nevertheless there is a drawback since the universal approximation theorem does not restrict the width of the network. This means that it eventually happens that the number of neurons in the hidden layer gets very large to approximate another hypothesis sufficiently good.

4.4 Backpropagation

Until now feedforward neural networks are discussed from a theoretical point of view. However we encountered the weights and biases of each layer as the parameters to train and the question now is how to efficiently update them accordingly to our training samples. The answer is clearly an iterative numerical optimizer. The good point is that they are already introduced during the discussion of the logistic regression method and it is possible to use them for neural networks too. The problem is that the calculation of the gradient of the loss function for all weights and biases is not as straight forward as in the logistic regression case.

However, to introduce the idea of the backpropagation algorithm we start with a very simple example. Consider a neural network with just one neuron in all layers. This means that the output of the last layer is given by

$$\hat{y} = a^{(L)} := \sigma^{(L)}(z^{(L)}), \tag{4.8}$$

where $z^{(L)} := \omega^{(L)} a^{(L-1)} + b^{(L)}$. Now it is possible to calculate the gradient of the loss function with respect to this weight for one training sample by using the chain rule:

$$\frac{\partial L_k}{\partial \omega^{(L)}} = \frac{\partial z^{(L)}}{\partial \omega^{(L)}} \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial L_k}{\partial a^{(L)}} = \frac{\partial z^{(L)}}{\partial \omega^{(L)}} \delta_k^{(L)}, \tag{4.9}$$

where $L_k := L(a^{(T)}(x_k), y_k)$.

The derivative of the weight of layer L-1 can be built on the derivative of the layer L:

$$\frac{\partial L_k}{\partial \omega^{(L-1)}} = \frac{\partial z^{(L-1)}}{\partial \omega^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \frac{\partial z^{(L)}}{\partial a^{(L-1)}} \delta^{(L)} = \frac{\partial z^{(L-1)}}{\partial \omega^{(L-1)}} \frac{\partial a^{(L-1)}}{\partial z^{(L-1)}} \omega^{(L)} \delta_k^{(L)} = \frac{\partial z^{(L-1)}}{\partial \omega^{(L-1)}} \delta_k^{(L-1)}. \quad (4.10)$$

One can see that going deeper into the network is nothing else than applying the chain rule for differentiation. To calculate the gradient with respect to the weight of layer l, one first needs to calculate the following error term by backward iteration:

$$\delta_k^{(l)} = \delta_k^{(l+1)} \omega^{(l+1)} \frac{\partial a^{(l)}}{\partial z^{(l)}} \text{ for } l = L-1, ...1 \quad (4.11)$$

Finally the gradient of the loss function with respect to the weight or the bias of layer l is obtained by the following equations:

$$\frac{\partial L_k}{\partial \omega^{(l)}} = \delta_k^{(l)} \frac{\partial z^{(l)}}{\partial \omega^{(l)}}, \quad (4.12)$$

$$\frac{\partial L_k}{\partial b^{(l)}} = \delta_k^{(l)} \frac{\partial z^{(l)}}{\partial b^{(l)}}. \quad (4.13)$$

Equations (4.12) and (4.13) describe the gradient for a specific weight for the training sample k when each layer of the network has just one neuron. In practice one usually has multiple neurons at each layer and a large amount of training samples.
However the first point is essentially already covered by the given equation since the inner derivatives are the same. In detail if one wants to analyze the contribution of the j-th neuron in layer l-1 to the i-th neuron in layer l it is necessary to calculate the derivative with respect to the weight $\omega_{ij}^{(l)}$ since this is connecting the two neurons.

In summary the following iteration scheme calculates the derivative resulting from the chain rule with respect to a specific weight or bias for a feedforward neural network with an arbitrary number of neurons at each layer:

$$\delta_k^{(l)} = \delta_k^{(l+1)} W^{(l+1)} \frac{\partial a^{(l)}}{\partial z^{(l)}} \text{ for } l = L-1, ...1, \quad (4.14)$$

$$\delta_k^{(L)} = \frac{\partial a^{(L)}}{\partial z^{(L)}} \frac{\partial L_k}{\partial a^{(L)}}. \quad (4.15)$$

One can see that the only difference to the one neuron neural network is that $\omega^{(l)}$ is replaces by the weight matrices $W^{(l)}$. Finally the gradients of interest are obtained by the following formulas [61]:

$$\frac{\partial L_k}{\partial \omega_{ij}^{(l)}} = \delta_k^{(l)} \frac{\partial z^{(l)}}{\partial \omega_{ij}^{(l)}}, \quad (4.16)$$

$$\frac{\partial L_k}{\partial b_i^{(L)}} = \delta_k^{(l)} \frac{\partial z^{(l)}}{\partial b_i^{(L)}}. \quad (4.17)$$

It is still left that the true gradient is the average of the gradients to all training samples. However during the discussion of the gradient descent method it is stated that calculating the full gradient is eventually very expensive. This is why a neural network is usually trained cumulative by just updating the weights and biases with respect to a small part of the training samples at once. This small part of the training samples are commonly referred as mini batches and this method is comparable to the stochastic gradient descent method.

In practice one usual starts the training of neural network by shuffling the training data. Afterwards the data set is split into mini batches where the size of a mini batch $\tilde{m}$ is much smaller than the number of total training samples. Furthermore the gradient needed for the backpropagation algorithm is then approximated by:

$$\nabla_{\vec{W},\vec{b}}\mathcal{L} \approx \frac{1}{\tilde{m}} \sum_{k=1}^{\tilde{m}} \nabla_{\vec{W},\vec{b}} L_k, \tag{4.18}$$

where $\vec{W} = (W^{(1)}, ..., W^{(L)})$ and $\vec{b} = (b^{(1)}, ..., b^{(L)})$.

Although the backpropagation algorithm provides a very descent way to train a neural network, it is very inefficient if the optimization possibilities are not exploited. This means that simply applying the formulas (4.16) to (4.18) in a serial manner would lead to a lot of redundant calculations because the inner derivatives reoccur for different steps.

Automated Differentiation

The solution for this is dynamic programming with automatic differentiation [45]. The main idea of dynamic programming is that results can be reused. So if a solution of specific inner derivative is expected to be part for another calculation, one simply saves the result for the later use. In addition automatic differentiation exploits the fact that a computer program just executes primitive operations at the lowest level. This means that it is possible to calculate the derivatives of complex functions as a combination of derivatives of primitive operations.

To outline the algorithm consider the simple neural network given by:

$$\begin{aligned} R_{(\mathbb{I},ReLU)}(\Phi)\left((x_1,x_2)^T\right) &= \mathbb{I}\left(\omega_3\left((\omega_1,\omega_2)^T ReLU\left((x_1,x2)^T\right)+b_1\right)+b_2\right) = \\ &= \omega_3\left((\omega_1,\omega_2)^T ReLU\left((x_1,x2)^T\right)\right). \end{aligned} \tag{4.19}$$

For the sake of simplicity the biases is set to zero because it does not change the conception of the algorithm. A computer evaluates the corresponding implementation of equation (4.19) by moving through a computational graph where each edge is just an operation on primitive variables.
Figure 4.1 shows a schematic computational graph of the neural network of equation (4.19). One can see that the calculation of the network is a composition of operations on primitive variables and evaluations of user defined functions, namely the ReLU activation function. This holds also for complex neural networks and other arbitrary functions. However the important thing to note here is that also the derivatives of the neural network consist only of these operations because of the chain rule for differentiation. The main idea of automated differentiation is now that the derivatives with respect to the input variables can be calculated concurrently with the evaluation of the function. In the context of backpropagation this means that one can evaluate the gradient on the fly when calculating the loss for training samples. In addition this can be highly parallelized since the computational graph of the function itself as well as for its gradients have a lot of independent calculation steps.

Table 4.1 shows that calculation steps of the example neural network (4.19) and its derivatives with respect to the weights. The first column of the table describes the steps for the neural network itself. Furthermore the second column contains the calculation steps for the derivatives.

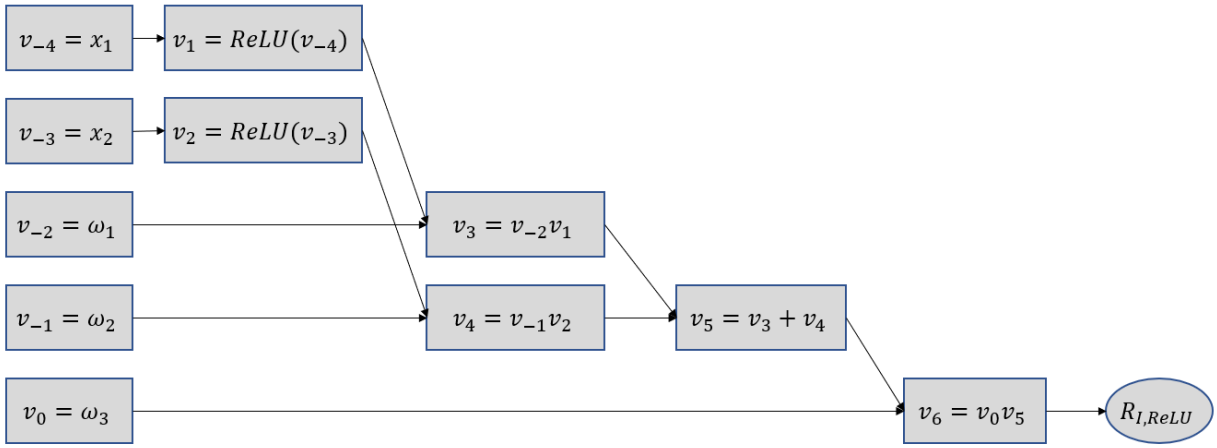

Figure 4.1: Sketch of the computational graph of the neural network given by (4.19).

The last three columns are the evaluated values of the second column. One can see that the calculations of the derivatives are independent from each other and some of the steps even do not need a value of the true function. This means that the determination of the gradient with respect to all weights can be done concurrently.
However on standard CPUs there is a limited amount of concurrent threads which means that just a small part of parallelization can be exploited. This is the main reason why neural networks are usually trained on GPUs and the efficient implementation of the backpropagation algorithm with automated differentiation build the basis of the nowadays success of neural networks [54].

$\boldsymbol{R}$			$\boldsymbol{\partial_{\omega_i}}$			$\boldsymbol{\partial_{\omega_1}}$			$\boldsymbol{\partial_{\omega_2}}$			$\boldsymbol{\partial_{\omega_3}}$		
v_{-4}	$=$	x_1	$\dot{v}_{-4}$	$=$	0	$\dot{v}_{-4}$	$=$	0	$\dot{v}_{-4}$	$=$	0	$\dot{v}_{-4}$	$=$	0
v_{-3}	$=$	x_2	$\dot{v}_{-3}$	$=$	0	$\dot{v}_{-3}$	$=$	0	$\dot{v}_{-3}$	$=$	0	$\dot{v}_{-3}$	$=$	0
v_{-2}	$=$	ω_1	$\dot{v}_{-2}$	$=$	δ_{i1}	$\dot{v}_{-2}$	$=$	1	$\dot{v}_{-2}$	$=$	0	$\dot{v}_{-2}$	$=$	0
v_{-1}	$=$	ω_2	$\dot{v}_{-1}$	$=$	δ_{i2}	$\dot{v}_{-1}$	$=$	0	$\dot{v}_{-1}$	$=$	1	$\dot{v}_{-1}$	$=$	0
v_{-0}	$=$	ω_3	$\dot{v}_{-0}$	$=$	δ_{i3}	$\dot{v}_{-0}$	$=$	0	$\dot{v}_{-0}$	$=$	0	$\dot{v}_{-0}$	$=$	1
v_1	$=$	$ReLU(v_{-4})$	$\dot{v}_1$	$=$	$ReLU'(v_{-4})\dot{v}_{-4}$	$\dot{v}_1$	$=$	0	$\dot{v}_1$	$=$	0	$\dot{v}_1$	$=$	0
v_2	$=$	$ReLU(v_{-3})$	$\dot{v}_2$	$=$	$ReLU'(v_{-3})\dot{v}_{-3}$	$\dot{v}_2$	$=$	0	$\dot{v}_2$	$=$	0	$\dot{v}_2$	$=$	0
v_3	$=$	$v_{-2}v_1$	$\dot{v}_3$	$=$	$\dot{v}_{-2}v_1 + v_{-2}\dot{v}_1$	$\dot{v}_3$	$=$	v_1	$\dot{v}_3$	$=$	0	$\dot{v}_3$	$=$	0
v_4	$=$	$v_{-1}v_2$	$\dot{v}_4$	$=$	$\dot{v}_{-1}v_2 + v_{-1}\dot{v}_2$	$\dot{v}_4$	$=$	0	$\dot{v}_4$	$=$	v_2	$\dot{v}_4$	$=$	0
v_5	$=$	$v_3 + v_4$	$\dot{v}_5$	$=$	$\dot{v}_3 + \dot{v}_4$	$\dot{v}_5$	$=$	$\dot{v}_3$	$\dot{v}_5$	$=$	$\dot{v}_4$	$\dot{v}_5$	$=$	0
v_6	$=$	v_0v_5	$\dot{v}_6$	$=$	$\dot{v}_0v_5 + v_0\dot{v}_5$	$\dot{v}_6$	$=$	$v_0\dot{v}_5$	$\dot{v}_6$	$=$	$v_0\dot{v}_5$	$\dot{v}_6$	$=$	v_5

Table 4.1: Evaluated table for automatic differentiation of the neural network given by (4.19).

The outlined way of automated differentiation is called the forward mode. It is also possible to calculate the derivatives by moving through the computational graph in a reversed order which is named the backward mode. However a detailed discussion of both modes as well as an implementation guideline can be found in [45]. Furthermore it is worth to mention that there are various libraries available which implement automatic differentiation very efficient for the common programming languages. In addition deep learning frameworks like Tensorflow 2 use this method during the training of neural networks.

4.4.1 Algorithm

Finally it is possible to pseudo code the backpropagation algorithm with automated differentiation by using the discussed theory.
The algorithm starts by initiating the optimizer with user defined hyperparamters. To give an example, a typical hyperparamter of gradient based algorithms is the step length. Furthermore it shuffles the training data and splits it into the mini batches. Afterwards it starts the loop where one iteration is usually referred as an epoch when talking about training a neural network. Within each epoch, it uses the optimizer to update the weights and biases for each mini batch in an arbitrary order. The calculation of the gradient is done with automated differentiation. Finally it returns the weights and biases after the last epoch is proceeded.

Algorithm 4 Backpropagation with Automated Differentiation

Inputs:
L; X; $\vec{W}_0$; $\vec{b}_0$; Optimizer; epochs, batch_size > 0
Initialize:
$\vec{W} \leftarrow \vec{W}_0$
$\vec{b} \leftarrow \vec{b}_0$
opt $\leftarrow$ Optimizer(params) (e.g. SGD, Adam, Nadam)
X = shuffle(X).reshape(-1, batch_size)
for epoch $= 1$ to epochs **do**
    for x in shuffle(X) **do**
        $\nabla_{\vec{W},\vec{b}} L \leftarrow$ automated differentiation
        $\vec{W}, \vec{b} =$ opt$(\vec{W}, \vec{b}, \nabla_{\vec{W},\vec{b}} L)$
    end for
end for
return $\vec{W}, \vec{b}$

To recap, the inner for loop as well as the calculation of the gradient by automated differentiation can be parallelized very well on GPUs. However when using a deep learning framework one usually does not have to optimize this algorithm since the software packages have already a very efficient implementation for that.

Nevertheless it is necessary to note that the loss surface of deep neural network is usually not convex. This means that gradient descent based methods can be trapped in local minima and this drawback is inherited by the the backpropagation algorithm. In total the initial values of the weights and biases as well as the hyperparameters of the optimizer are very important to successfully train the network by backpropagation [42].

4.5 Autoencoders

There are various fields where dimensionality reduction is necessary to develop efficient solutions for complex problems. To motivate that technique in the context of micromagnetism, it is assumed that the goal is to calculate the magnetization reversal process of a magnet consisting of millions of small magnetic grains. This is a realistic case since we already stated that the length of interest of magnets in applications is in the range of millimeters while the grain sizes of hot-deformed magnets are usually tenths to hundreds of nanometers. However the representation of the magnetization vector of a multigrain has the dimension of the number of grains times the three spatial coordinates. This essentially means that a predictor based on neural networks operating in the real space would have millions of weights and biases within each layer which is

computationally intractable. The solution for that problem is dimensionality reduction. However standard techniques like principal component analysis have the drawback that they can not conserve nonlinear dependencies. This is not the case for feedforward neural networks as long as the activation function is not linear as known from the universal approximation theorem.

Formally an autoencoder is an unsupervised neural network which tries to copy the input to its output [61]. However the interesting part is that there is one central intermediate layer which has a different dimension than the input layer. This technique is commonly used for dimensionality reduction or feature extraction. The focus of this work lays on the first case where the intermediate layer compresses the data to the most important features which are necessary to obtain the original state, assuming that the method is successfully reducing the dimension of the underlying problem.
An autoencoder consists of two parts, namely the encoder and the decoder. The encoder maps the input data to a latent space where the dimension of this space is defined by the central intermediate layer. In addition the decoder is able to decode the latent space representation of the input back to the real space. Since the representation in the latent space is unknown upfront, both parts of the autoencoder are trained together where the loss is usually calculated in the real space and the weights and biases of both models are updated at once. The following equation describes the empirical risk of an autoencoder:

$$\mathcal{L}_{AE} = \frac{1}{m} \sum_{k=1}^{m} L(D(E(x_k)), x_k), \tag{4.20}$$

where $E : \Omega \to \mathcal{F}_\Omega$ is the encoder and $D : \mathcal{F}_\Omega \to \Omega$ is the decoder. In addition, the space $\mathcal{F}_\Omega \subset R^k$ is the latent space. When using this technique for dimensionality reduction, the dimension of the latent space is lower than the dimension of the real space. In that case one speaks of an undercomplete autoencoder. On the other hand an overcomplete autoencoder would have a latent space of higher dimension than the original space. However this work focuses on undercomplete autoencoder since we are mainly interested in dimensionality reduction.

Before going into the details of the model, it is worth to mention that eventually the autoencoder happens to just learns the identity which does not seem to be very useful. This is the case if the dimension of the latent space is equivalent to the dimension of the input space. Nevertheless it can also happen if the number of weights and biases is high enough to memorize every training sample [57]. This causes a perfect accuracy on the training set but it would lead to overfitting which eventually introduces large errors on unseen data. However methods like cross validation and a sufficiently large set of training samples should avoid this problem in the practical use.

4.5.1 Convolutional Layers

So far feedforward neural networks are discussed in a way where dense layers are used as the blue print of hidden layers. To recap, a dense layer is fully connected to the previous layer which means that each neuron receives an input of every neuron of the layer before. However real world scenarios might have a intrinsic geometry which allows the use of more sophisticated layer types. The idea of convolutional layers comes from image recognition. Without going into details, the goal of image recognition is to identify objects within an image. From a human perspective this is an intuitive task but on a computer an image is nothing else than a pixel map. In other words a digital image is a two dimensional matrix where each entry is a three dimensional byte array, assuming that standard red-green-blue (RGB) is in use. Furthermore the indices of the

matrix define the position of the pixel. The idea is now that the objects to detect on the image eventually have a specific region where they occur.
To give a vivid example, it is assumed that the algorithm should identify people on a picture of the last semester graduates. This is eventually a tough task to solve if the machine learning model needs to proceed the complete picture at once. However one can start by detecting small characteristic features like the eyes and mouths of the people before trying to identify the complete human. These features occur in just a small region of the image and therefore a dense layer might not be the sufficient layer type to choose.
Convolutional layers restrict the input for each neuron to just a few neurons of the last layer which are geometrically nearby each other. In addition they usually concatenate these inputs since the goal is to reduce the dimension. As a consequence the total number of trainable parameters is drastically reduced. This technique allows to identify local characteristics of the image at the first layers while global characteristics are recognized within the deeper layers of the network. Although this method was initially developed for image recognition, it also shows its strength when applied to physical problems because local interactions eventually have an other impact to the system than global ones as we have seen in chapter 1.

A rigorous mathematical definition of a convolutional layer is hard to obtain since they are valid for different input dimensions and there are refined elements for specific problems. For the context of this work, it is just defined for two input dimensions and without a stride.
However the central element of this layer type is a convolution. In terms of deep learning, this is formally defined by:

Definition 11. *Let $X, Y, \in R^{n\times n}$. The convolution $Z = X * Y$ is defined as:*

$$Z[i,j] = \sum_{k,l=0}^{n-1} X[i-k, j-l]Y[k,l],$$

where zero-padding is used if $i-k$ or $j-l$ is less than zero or greater than $n-1$.

Given that definition of a convolution, a convolutional node can be stated as the follows:

Definition 12. *Given $X \in R^{N_1\times N_2\times S}$, $W \in R^{F\times F\times S}$ and $b \in \mathbb{R}$. A convolutional node computes:*

$$Z = W *_{cnn} X + b,$$
$$W *_{cnn} X := \sum_{k=1}^{S} X[:,:,k] * W[:,:,k] + b.$$

where W is named the filter or kernel.

Finally it is possible to define the convolutional layer by the following [66]:

Definition 13. *Given $X \in R^{N_1\times N_2\times S}$ and $((W_i, b_i))_{i=1}^{K} \subset \mathbb{R}^{F_1\times F_2\times S} \times \mathbb{R}$ and a activation function $\sigma : \mathbb{R} \to R$. A convolutional layer calculates:*

$$A = \sigma(Z[:,:,i]) : = \sigma(W_i *_{cnn} X + b_i),$$

where σ is applied component-wise and $Z \in R^{N_1\times N_2\times K}$.

One can see that a convolutional layer can be defined in an equivalent way of a standard dense layer where the matrix vector product is replaced by the convolution. Nonetheless there are some elements of the definitions which need to be discussed before moving on to an practical example.

The first thing to note is that the input to a node of the convolutional layer is restricted to the F_1 times F_2 neighboring nodes of the previous layer. In addition the filters given by the matrices W_i are applied to all input channels of the layer and the depth of the output image is the equal to the number of filters applied within the layer [12]. This essentially means that a convolutional layer transforms a volume of values into another volume of values which eventually has a different dimension. Finally the activation function is applied component-wisely. However one can clearly see that the strength of a convolutional layer is highly depending on the filters and the question is how to define these.

Lets assume that there is an image with a box in the middle and the goal of the algorithm is to recognize this object. For the sake of simplicity the image is given as two dimensional matrix of integer values. A human perspective approach is now to first define a filter which recognizes the boarders of the box and later on having a layer which classifies a box out of four boarders. Using this approach one can choose the following filter for the first step:

$$W_{edge} = \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} \tag{4.21}$$

Furthermore applying this filter to the schematic image leads to the following result:

$$\begin{pmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} * \begin{pmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{pmatrix} = \begin{pmatrix} -1 & -2 & -3 & -3 & -2 & -1 \\ -2 & 5 & 3 & 3 & 5 & -2 \\ -3 & 3 & 0 & 0 & 3 & -3 \\ -3 & 3 & 0 & 0 & 3 & -3 \\ -2 & 5 & 3 & 3 & 5 & -2 \\ -1 & -2 & -3 & -3 & -2 & -1 \end{pmatrix} \tag{4.22}$$

The calculation shows that the values for the edges of the box are significantly greater than the other values after the filter is applied to the digital image. In addition if the ReLU activation function is used in that example the values of the boarders are the only values which are unequal to zero. As mentioned before, the goal of the algorithm can now be achieved by adding another layer on top of the convolutional layer which does the classification.
However defining an edge detection filter is a solvable task but the engineering of generalizing filters for various different problems is usually very difficult from a human perspective. That is the point where machine learning kicks in, since convolutional layers are trained in the same fashion as the dense layers. In detail the filters or kernels are adjusted to identify specific characteristics of the underlying problem. This means that the weights and biases of the filters are updated during the training of the model.

Convolutional layers were initially applied to image recognition in two dimension. Furthermore the defined version of this layer type does not include downsampling. Therefore different layers types are in use. Typically these downsampling layers consist out of fixed operation and they are commonly referred as pooling layers. There are different types of pooling layers but they usually concatenate the inputs to single values. A very famous example of a pooling layer is called Max-Pooling. This does nothing else than just keeping the larges value of the inputs and removing the others [50].
Nowadays convolutional layers are also applied in different dimensions since they outperform standard layers for specific problems. In addition they can include downsampling directly within the layer by using a stride. The definition 13 misses this very important hyperparameter because it is implicitly assumed that it is set to one. In detail this value defines how far the filter is moved

from one calculation to the next. For example a stride of two means that the filter is shifted by two units. This procedure reduces the dimensionality already within the convolutional layer because the number of elements in the output matrix is roughly the number of total input values divided by the stride in each dimension and for each filter. The main benefit of this procedure is that it introduces trainable paramters for the dimensionality reduction step. This is not the case for pooling layers since they are consisting of fixed operation. Nevertheless there is a drawback because the increased number of weights and biases require more computational power to be trained. However recent researches show that there are applications where convolutional layers with strides outperform the classical setup with pooling layers [10].

4.5.2 Convolutional Autoencoders

Convolutional autoencoders are machine learning models based convolutional neural networks. Nonetheless it is worth to mention that they include dense layers too.
Using this concept, one usually builds the encoder by start stacking convolutional layers. Normally the dimension is already reduced in first layers by adding a stride or pooling layers. Furthermore there are dense layers in use when getting towards the latent space. The decoder reverses this process by first enlarging the dimension with dense layers and later on using transposed convolutional layers to get back the dimension of the real space. A transposed convolutional layer is the inverse of a convolutional layer which essentially means that it can decode the features of the input [69].

Figure 4.2 shows an example architecture of an autoencoder implemented with Tensorflow 2 and Keras. The encoder successively reduces the dimension of the three dimensional input. In detail each convolutional layers uses a stride of two in all direction which descales the input by a factor of 2^3 in each layer. Furthermore the architecture uses four filters in the first convolutional layer and two in the second. When reaching the dimension of $(2, 2, 2) \times 2$ a flatten layer is applied to transform the input to the correct dimension for the stacked dense layer. Finally the input is downsampled to 8 float values which represent the embedded image in the latent space.
The decoder reverses that process by first transforming the representation of the latent space into a dense layer with twice as many neurons. Afterwards the output of the dense layer is reshaped in a way that it can be used as an input for the transposed convolutional layers. Finally two transposed convolutional layers are added to convert the image back to the real space.

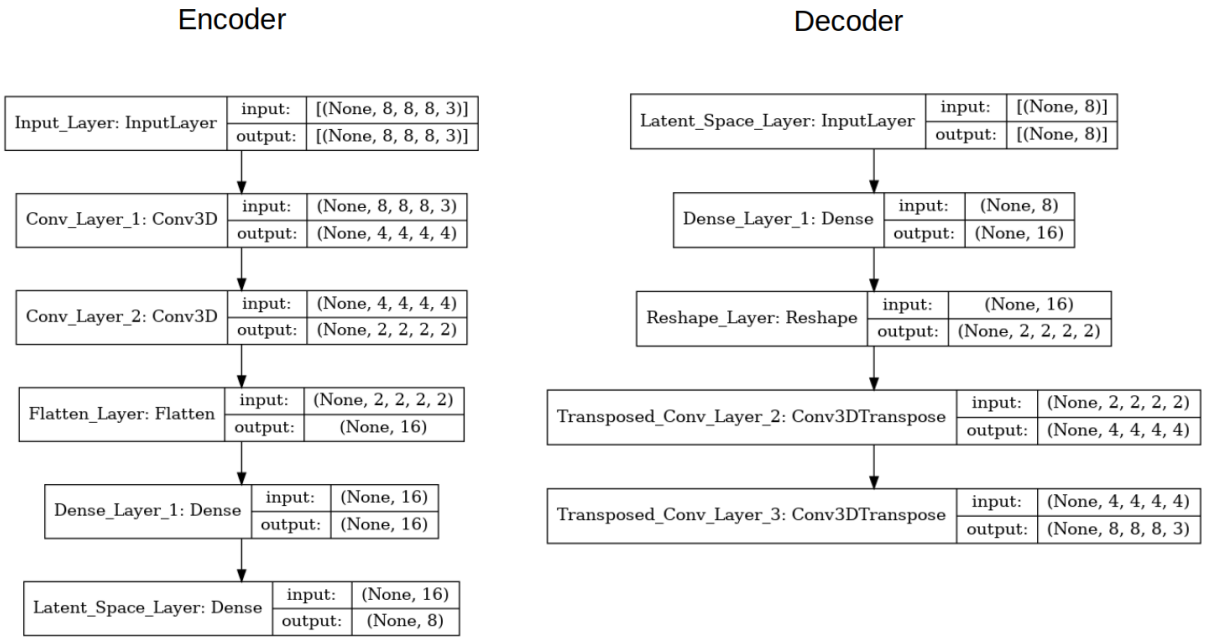

Figure 4.2: Example architecture of autoencoder.

Chapter 5

Machine Learning applied to Magnetization Reversal Processes

The following chapter combines the theory of the previous chapters by solving a physical problem with machine learning. It starts by encoding the magnetization patterns which occur during the magnetization reversal. This enables to reduce the dimensionality of the underlying problem. Afterwards it describes predictors for the complete demagnetization curve of an unseen microstructure.
However the microstructures used for the experiments are very simple and do not represent real world examples. Nevertheless they are used as a proof of concept of this method since it is applied in a way that it can be adopted to systems of larger scale. That is why the model for estimating the hysteresis curve is operating on the latent space of the autoencoder although this is eventually not necessary for the given microstructures.

5.1 Introduction

During the theoretical introduction to machine learning we encountered the necessary elements to apply this method successfully. To recap, the starting point is a data set which was drawn from the underlying distribution of the model. Furthermore the next step is to define the hypothesis spaces or in simpler words the type of machine learning models which are used for the task. In addition one has to find an appropriate loss function which describes the error of the models in physical relevance. Finally the hyperparameters of the model and the optimizer must be evaluated by using techniques like cross validation.

In general the underlying problem of the work is the magnetization reversal process of a multigrain microstructure. This means that an autoencoder is trained to represent the magnetization state at each external field step of the hysteresis curve in the according latent space. However it is required that the decoding of the embedded magnetization state must reproduce the original image reasonably well. In addition the predictors estimate the magnetization in the latent space at a new field step by only taking information from previous field steps.
All models are based on neural networks during this work. In detail the autoencoders rely on convolutional neural networks while the predictors are fully connected feedforward neural networks. The test cases include different types of architectures and try to evaluate the best set up for this use case.
Furthermore the implementations of the models are done with the machine learning framework Tensorflow 2 used in Python 3.8. This is an open source software package which was initially developed by Google and it provides a set of software tools to efficiently use machine learning [1]. However the used methods rely only on the packages related to deep learning with neural

networks. In addition there is the high level API Keras which simplifies the use of this type of models in Tensorflow 2. This programming interface provides an intuitive access to deep learning from a human perspective. However it is worth to mention that some functionalities of Tensorflow 2 need to be explicitly used without Keras but they are not required within this work.

Before proceeding the experiments, it is necessary to describe the data set in use. The complete set consists of simulated data without practical experiments. The microstructures are generated by Neper which is a tool for generating polycrystalline structures and meshing. In detail each structure is three dimensional and it consists out of 64 cubic grains aligned uniform in all directions. Furthermore all grains have the same intrinsic magnetic properties besides their easy axes. The easy axes are randomly defined within an angle of 35 degree between it and the z-coordinate. Furthermore the reduced order model is used to simulate the magnetization reversal process of the microstructures. In detail the external field is varied from 8 to -8 Tesla in 1000 steps. This means that the external field is reduced by -0.016 Tesla at each field step. The complete data set consists out of 10 full simulation which leads to 10 000 different magnetization states. For each full simulation a different random distribution of the easy axes is used. From this set nine simulations are used for training while one simulation is the test set.
Figure 5.1 shows the simulated magnetization reversal process of one microstructure. Furthermore there are four magnetization states of the structure visualized explicitly. To recap, this simulation corresponds to 1000 samples of the data set.

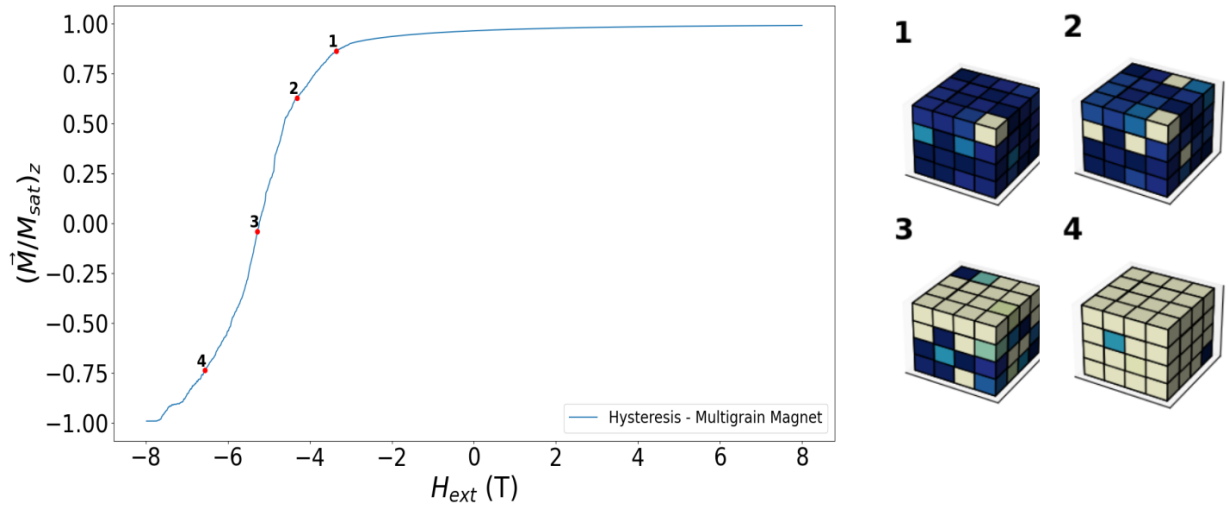

Figure 5.1: Simulated magnetization reversal process of multigrain magnet, color shows the magnetization along the z-axis.

5.2 Autoencoder

The autoencoder is developed to reduce the dimensionality of the problem. In detail the encoder transforms the original magnetization state to an embedded image with a lower dimension and the decoder is able to reproduce the original image from that latent space representation. Here the term image refers to one state of magnetization. In addition one state of magnetization of the complete microstructure is a matrix with the dimension $(4 \times 4 \times 4) \times 3$. This means that there are 64 entries of three dimensional arrays which describe the magnetization of each grain along the x-, y- and z-axis. This can be understood as a three dimensional image where the RGB values are replaced by the components of the magnetization.

Furthermore each state of magnetization calculated by the reduced order model is a single sample within the data set. In summary this means that the training set contains 9000. Furthermore the test set consists out of 1000 samples of a different simulation.

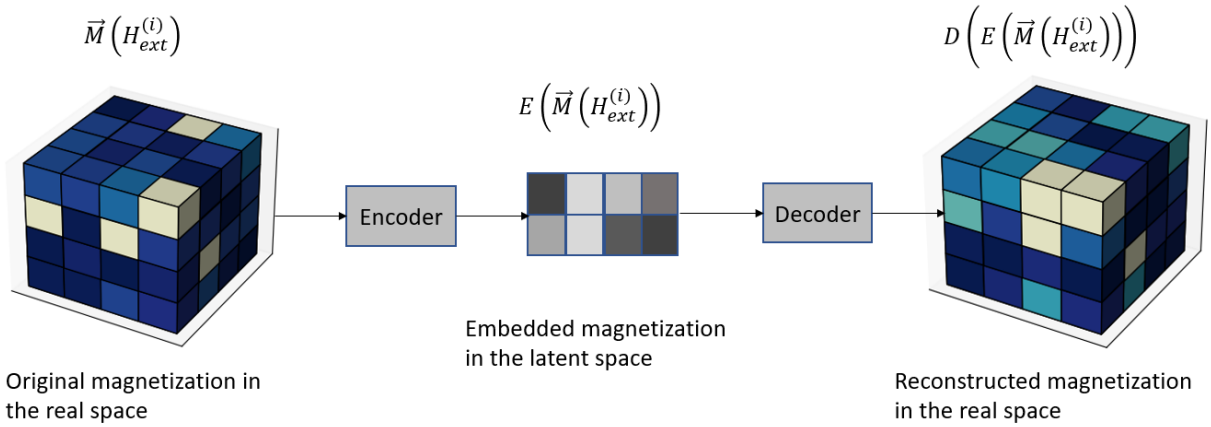

Figure 5.2: Schematic overview of the reconstruction of one magnetization state by using the autoencoder. Reconstructing a complete hysteresis requires to apply this method to every state along the demagnetization curve.

5.2.1 Loss Function Engineering

The first question to answer is how to define an appropriate empirical risk minimization problem for the task to solve. Since this is regression problem, the standard mean squared error is a reasonable loss function to choose. In this case this error is calculated as the sum of squared differences of each entry of the magnetization vector.
However micromagnetism also states that the length of the magnetization vector must be constant. This implies that the loss function can be enriched by a term which takes care of the preservation of the norm. Using these ideas, one can formulate the candidate loss function by the following equation [38]:

$$L_{AE}(\hat{x}, x) = a_1 \underbrace{\sum_{i,j,k=1}^{4} \sum_{l=1}^{3} (x_{ijkl} - \hat{x}_{ijkl})^2}_{L_{MSE}} + a_2 \underbrace{\sum_{i,j,k=1}^{4} \left(1 - \sqrt{\sum_{l=1}^{3} \hat{x}_{ijkl}^2}\right)^2}_{L_{Norm}}, \tag{5.1}$$

where $\hat{x}$ is the reproduced image in the real space of the autoencoder. The indices i, j and k number the grains on a regular geometric grid. Furthermore l describes the magnetization component. Therefore x_{ijkl} defines the l-th component of the unit vector of the magnetization in grain ijk.
The first term of the right hand side of equation (5.1) defines the mean squared error for this example. The second term of this formulation penalizes the model if the reproduced image does not conserve the norm. Furthermore the parameters a_1 and a_2 describe the relative weights of the terms since the model eventually produces better results if they are not uniformly weighted. Furthermore it is worth to note that the loss function only accounts the means squared error if the parameter a_2 is set to zero.

Model Architecture

However to test which configuration of the loss function provides the better results, one has to do numerical experiments with an example model. For this case the autoencoder is a convolutional neural network with setup described in table 5.1.
Here the encoder consists of one convolutional layer with four filters where $F_1 = F_2 = 2$ and it uses a stride of 2. In addition there are three dense layers. The latent space consists of 16 neurons which means that the dimensionality is reduced by a factor of 12 since the image is described by 192 floats in the real space. The decoder applies the same setup of layers in the reversed order. In total this leads to 4983 trainable parameters.

Layer (type)	Output Shape	Param #
input (InputLayer)	[(None, 4, 4, 4, 3)]	0
conv3d (Conv3D)	(None, 2, 2, 2, 4)	100
flatten (Flatten)	(None, 32)	0
dense (Dense)	(None, 32)	1056
dense_1 (Dense)	(None, 32)	1056
dense_2 (Dense)	(None, 16)	528
dense_3 (Dense)	(None, 16)	272
dense_4 (Dense)	(None, 16)	272
dense_5 (Dense)	(None, 32)	544
dense_6 (Dense)	(None, 32)	1056
reshape (Reshape)	(None, 2, 2, 2, 4)	0
output (Conv3DTransp)	(None, 4, 4, 4, 3)	99
Total params: 4,983		
Trainable params: 4,983		
Non-trainable params: 0		

Table 5.1: Autoencoder architecture for testing the different loss functions.

Furthermore the activation function for all layers is the exponential linear unit [14] which is described by equation (4.6). In addition the kernels are initialized by the Tensorflow 2 function 'he_normal' which draws the initial weights from a truncated normal distribution around 0 where the standard deviation is related to the number input weights [1]. He et. al. have shown that this kernel initialization increases the models accuracy in their work [28] in 2015.

The training of the network is done by backpropagation where 20% of the training samples are used for validation. Furthermore one mini batch contains 64 training samples. Moreover the Nadam algorithm is used as the optimizer. The initial learning rate of the method is set to 0.0001. In addition this parameter of the optimizer is reduced during training by the Keras callback 'ReduceLROnPlateau' if the monitored validation loss stays constant though a certain amount of epochs. Additionally early stopping is used to avoid overfitting. This means that if the validation loss does not decrease significantly after some epochs, the training is stopped before reaching the maximal number of iterations through the training samples.

Learning Curves

The first analysis of the model is done by learning curves. However the loss given by equation (5.1) can vary in scale for different values of a_1 and a_2 because the terms eventually have a different intrinsic size. Therefore it is necessary to compare them independently to avoid misinterpretations of the results.

Anyhow before going into the details, it must be checked if machine learning is at all the appropriate method to tackle this type of problem. Moreover it is necessary to verify that the training set is rich enough to describe the underlying distribution and therefore the standard learning curve is the method to choose.

Figure 5.3 shows the loss of the model for different values for a_1 and a_2 in dependence to the number of epochs. One epoch is a complete iteration through the training data. Each graph represents either the values for the training set or the values for the validation set.
The plots show that the losses decrease with number of epochs. This is the expected behavior and it proves that the model is not underfitting. In addition the graphs indicate that the complexity of the model fits the problem since the overall losses on the validation sets are reasonably small after the training is finished. Furthermore the errors on the validation sets decrease until the last epoch. This indicates that the model is not overfitting for all of the used configurations.
Moreover using a smaller weight for the term, which penalizes the model for an incorrect norm, than the weight for the mean squared error introduces a larger total loss. This is eventually related to the fact that it is harder for the model to minimize the mean squared error than conserving the norm. As a consequence, adding the loss term of the norm decreases the overall loss. Furthermore setting the weight of norm error to 15% within the loss function reduces the number of epochs which are required to reach a plateau of the function to minimize. This is in general a figure of merit since it reduces the computational costs to train the model.

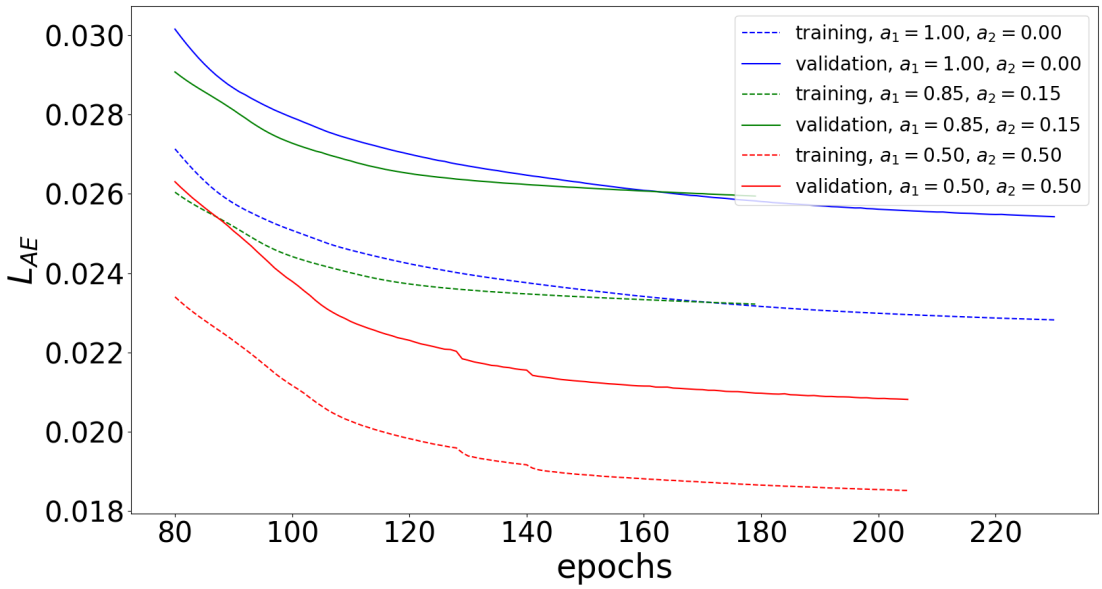

Figure 5.3: Loss of training set and validation set for different values of a_1 and a_2

However it is still needed to analyze how the mean squared error and the penalization of not preserving the norm interfere with each other. Therefore figure 5.4 shows the mean squared error dependent on the number of epochs for the different loss functions of the previous plots.
The graphs indicate that the model does not loose the accuracy in terms of the mean squared error by adding the penalization term. This means that including the norm in loss function reduces the mean squared error simultaneously. Nevertheless it is necessary to mention that this behavior is only shown if the norm related term is weighted smaller than the term which calculates the mean squared error. In numbers this means that setting the relative weights a_1 to 0.85 and a_2 to 0.15 enables the model to reach approximately the same minimum of the mean squared error by

less epochs. Nevertheless it is worth to note that weighting the norm error equally to the mean squared error introduces a significantly higher mean squared error.

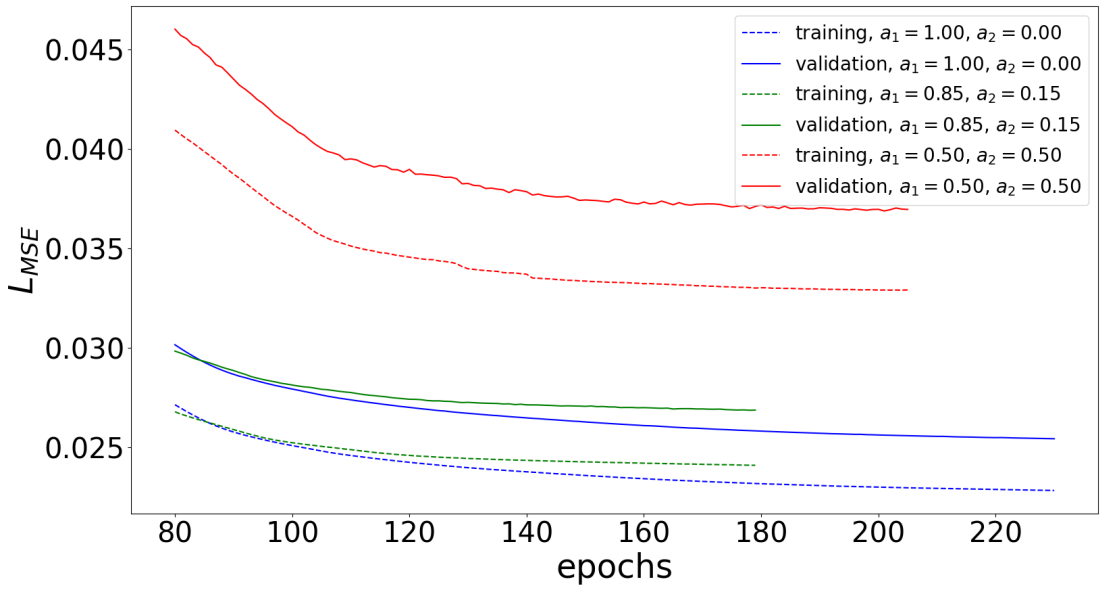

Figure 5.4: Mean squared error of training set and validation set for different values of a_1 and a_2

Finally figure 5.5 displays the norm error with respect to the number of epochs.
As expected, adding the physical information of the preservation of the norm to the loss function significantly reduces the error in the length of the predicted magnetization vectors. In addition the relative difference between the norm error on the training set and the norm error on the validation set decreases by enlarging the relative weight a_2. However it is necessary to note that the preservation of the norm is just additive information since this term does not relate the predicted magnetization to the ground truth. This means that weighting the norm error to high would lead bad predictions since the model would not learn the underlying data distribution in any sense.

In summary the learning curves show that adding information of the underlying physical theory to the loss function increases the overall performance of the autoencoder. In addition it significantly reduces the number of epochs needed to reach a minimum of the loss function. Furthermore the learning curves show that the loss landscape changes with the relative weights which is the expected behavior. However this leads to the fact that a qualitative comparison needs to analyze the terms separately.

Reproduced Hysteresis

The question stays how the model performs on an unseen test set containing a full magnetization reversal process for the different configurations. To recap, the learning curves contain the evaluation on validation sets randomly drawn out of the 9000 training samples but there is still a full simulation left for testing.
Figure 5.6 shows the magnetization reversal process of an example microstructure simulated by the reduce order model. In addition there are the plots of the reproduced magnetization of the structure where the autoencoder is trained with the different loss functions which are analyzed in the previous section. The graphs are created by predicting the magnetization state of all steps of the external field and plotting the z-component correspondingly.
In general the graphs indicate the used model is able to reproduce the demagnetization curve.

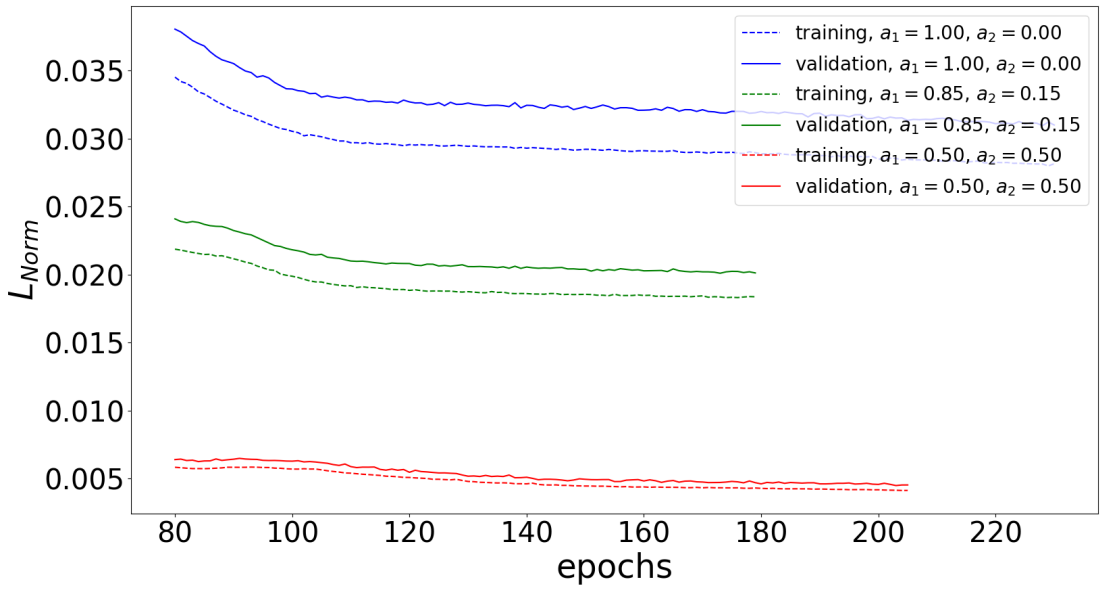

Figure 5.5: Norm error of training set and validation set for different values of a_1 and a_2

Furthermore one can see that the mean squared error is a good error measurement for the underlying problem. However there are differences in the performance related to the configuration of the loss function. In detail the plots show that using a model that minimizes a loss which equally weights the norm error and the mean squared error performs worse than a model which weights the mean squared error higher than the norm error. However weighting the norm conservation term by 15% provides a model with a reasonably good reconstruction. Here the main difference to prediction of the model without the norm penalization term occurs in the range from -3 to -6 Tesla.

Nonetheless it is worth to note that the model in this test case gets inputs with a normalized magnetization. So for this case the autoencoder predicts accurately if it does not change the norm but this is eventually not true if there is an additional machine learning model which predicts the magnetization state of the microstructure in the latent space. If the predictor produces embedded images which are not normalized in the real space, the autoencoder must be able to do this normalization. In total this is an additional argument to add the norm error to the loss function since the mean squared error on its own does not include normalization.

Finally the analysis shows that it is possible to reproduce the demagnetization curve with an autoencoder trained by the loss function given by equation (5.1). However the dominating term of the loss function must be the mean squared error since weighting the norm related term to high leads to worse reconstructions. Nevertheless the results indicate that including the norm error reduces the computational expense for training the model. Furthermore it eventually includes normalization capabilities to the autoencoder.

5.2.2 Architecture Optimization

A very important part of solving a problem with machine learning is the evaluation of the best hyperparameters of the model. To recap, hyperparamters are defining the model and they are not updated during the training process. Autoencoders based on neural networks have several different parameters to tune, for example the activation functions, the initialization of the weights and biases and the parameters for the optimizer. However during this work the focus for optimization lies on the depth of the network and the dimensionality of the latent space.

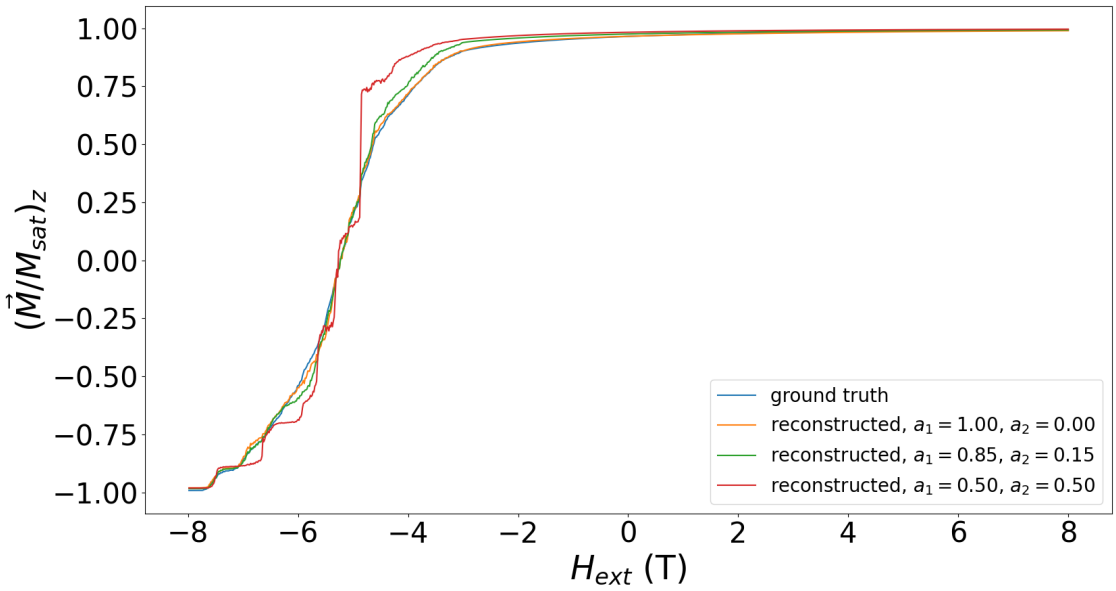

Figure 5.6: Full demagnetization curve of example microstructure. Ground truth labels the results of the reduced order model. Reconstructed images are the predictions of the autoencoder for different loss functions.

Model Architectures

There are various ways to design the architecture of the autoencoder. However deep networks have the benefit of more trainable parameters but they are more expensive to train. In addition the number of latent variables define the possible dimensionality reduction factor of the problem. To evaluate the best setup, it is necessary to test different architectures.

The first autoencoder is already in use for the previous experiments and it is given by table 5.1. This model is further referred as deep_16.
In addition the second model consists of the same number of layers as deep_16 but it has a latent space dimension of only 8. This means that the dense layer 'dense_3' is reduced to 8 neurons. The model is referred as deep_8. This architecture reduces the trainable weights to 4719.

Moreover there are two flatter autoencoders. The first one is named flat_8 and it is given by table 5.2. Here the convolutional layer is the same as for the deeper models but there is only one dense layer before and after the latent space. This leads to a total of 1807 trainable parameters.

The last model for testing is called flat_16 and it has the same layer structure as flat_8 but with only 16 latent space neurons. Therefore the shape of the middle layer is changed from 8 to 16. This model has 2327 trainable weights.

The training of the models is done in the same way as in section 5.2.1. The weights and biases are updated by backpropagation where the Nadam method is used as the optimizer. The initial step length of the algorithm is set to 0.0001 and it is reduced during training by the Keras callback 'ReduceLROnPlateau'. Moreover early stopping is used to avoid overfitting.

Layer (type)	Output Shape	Param #
input (InputLayer)	[(None, 4, 4, 4, 3)]	0
conv3d (Conv3D)	(None, 2, 2, 2, 4)	100
flatten (Flatten)	(None, 32)	0
dense (Dense)	(None, 32)	1056
dense_1 (Dense)	(None, 8)	264
dense_2 (Dense)	(None, 32)	288
reshape (Reshape)	(None, 2, 2, 2, 4)	0
output (Conv3DTransp)	(None, 4, 4, 4, 3)	99
Total params: 1,807		
Trainable params: 1,807		
Non-trainable params: 0		

Table 5.2: Model architecture of flat_8.

Furthermore the loss function defined by equation (5.1) is used where a_1 is set to 0.85 and a_2 is equal to 0.15. However the evaluation of the loss function shows that adding the norm error term reduces the accuracy of the model at external field values where most of the grains reverse their magnetization. This is eventually related to the fact that the number of training samples where grains are switching due to the increasing external field is less than the number of training samples where the external field does not switch grains. Looking back to figure 5.6, one can see that the grains are switching in the range of -8 to -3 Tesla because the magnetization of the complete microstructure does not change significantly before. This essentially means that only approximately 40 % of the training data contains grains with a reversed magnetization. To mitigate that problem, the training samples are weighted due to the change of the z-component of the magnetization state in relation to the previous external field step. The following formula provides the calculation scheme of the weights:

$$\begin{aligned} w_1 &= 0, \\ w_i &= \sum_{j=1}^{64} |m_{j,z}(H_{ext}^{(i)}) - m_{j,z}(H_{ext}^{(i-1)})| \text{ for } i = 2, ...1000. \end{aligned} \tag{5.2}$$

Here $m_{j,z}(H_{ext}^{(i)})$ denotes the magnetization along the z-axis of the j-th grain at the i-th external field step.
Equation (5.2) assigns larger weights to training samples which have a significant change in magnetization in relation to the previous field step. This further has the effect that the weights and biases of the models are more sensible to the training samples which are partially reversed. In total this should improve the performance of the models in the part of the demagnetization curve where the magnetization is reversed.

Learning Curves

Figure 5.7 plots the learning curves of the models given in the previous section. All graphs show the loss of the models on the training set or the validation set in relation to the number of iterations through the complete training data.
The first thing to note is that all models succeed in training since the loss on the training data as well as on the validation data is substantially reduce during the training. This means that all hypothesis spaces are complex enough to approximate the underlying data distribution. In addition the models do not tend to overfit since the loss on the validation sets are decreasing until the training is finished.
Another interesting thing to note here is that the deeper networks have approximately the same performance on the validation set independent on the number of latent space variables. This is

not the case for the flatter models since flat_16 outperforms flat_8 by far. This indicates that the number of trainable values of the flat autoencoder with only 8 latent space neurons is too low to approximate the underlying relation in more detail. However flat_16 shows the best overall performance. This is eventually related to the fact that the more complex models would need more training samples for more accurate results.

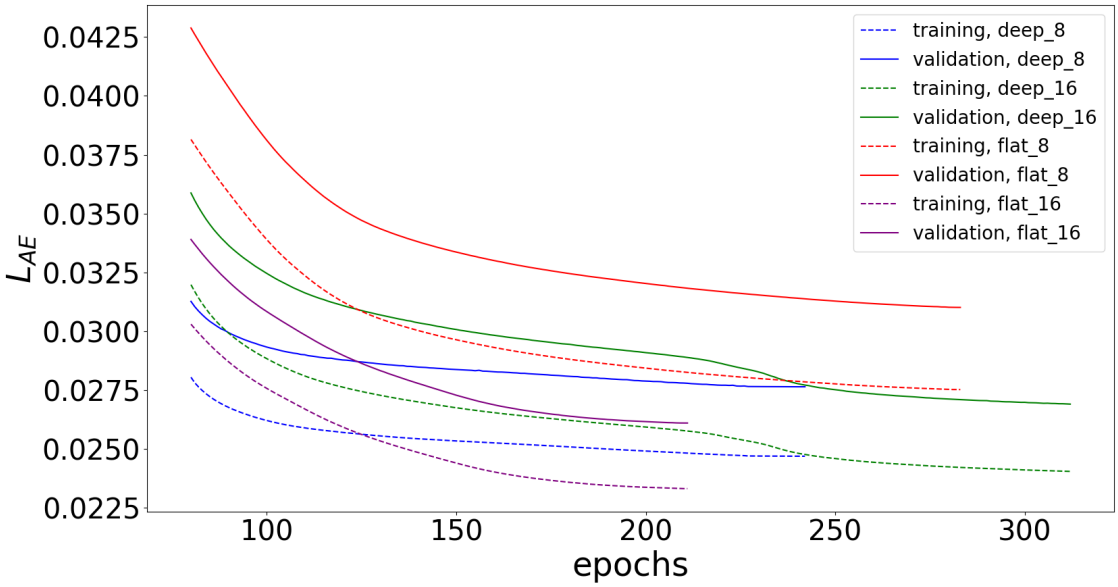

Figure 5.7: Learning curves of the models of section 5.2.2.

Demagnetization Curve

Figure 5.8 shows the demagnetization curve calculated by the reduced order model and the reproduced hysteresis of the autoencoders described in the section 5.2.2.
In general all models reproduce the magnetization states reasonably well. However there are slight differences in the range of -8 to -5.5 Tesla. Here the worst performance is shown by flat_8 which is clearly related to the fact that this model has the greatest overall loss after the training. In addition the autoencoders with a latent space dimension of 16 are more accurate than the models with the lower latent space dimension. In total flat_16 does the most accurate predictions but there is just a slight difference to deep_16.

In summary the predictions on the unseen test data replicate the findings of the learning curves. This shows that the autoencoders based on neural networks are able to reduce the dimensionality of the underlying problem since they are able to reproduce the magnetization in the real space reasonably well. However the deeper models eventually need more training samples to outperform the flat autoencoder with 16 latent space variables because they have more weights and biases. Finally one can say that flat_16 fits best for this setup since it produces the most accurate results.

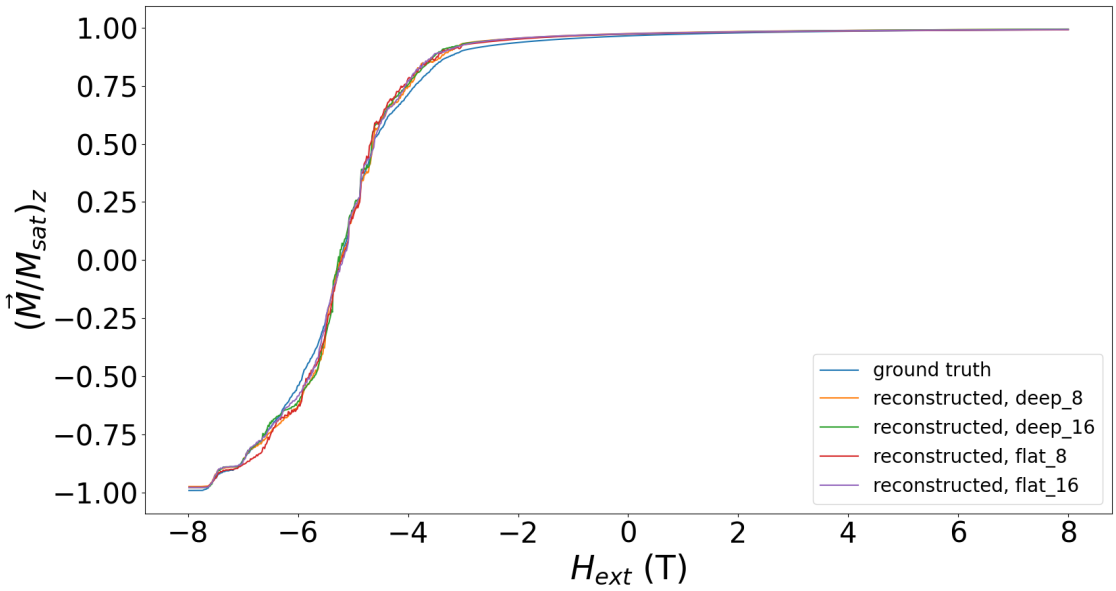

Figure 5.8: Full demagnetization curve of example microstructure. Ground truth labels the results of the reduced order model. Reproduced images are the predictions of the different autoencoders given in section 5.2.2.

Reproduced Magnetization States

Although the overall goal of the autoencoder is to reproduce the complete hysteresis of a microstructure, it is worth to look into the actual reproduced magnetization states of the models. Therefore plot 5.9 displays the magnetization state simulated by the reduced order model and the autoencoders for two different external fields. There are 3D plots of the microstructure where the color indicates the magnetization along the z-axis.
The first row of the plot shows that the autoencoders smooth the magnetization. This is a common behavior of this type of model [17]. For example looking to the upper side of the cubes of flat_16, one can see that the reproduced images indicate that grains are rotating further out of their preferred axis before the irreversible switch occurs. However comparing flat_16 to the ground truth, one can see that there are more grains rotating reversible while there are less grains which are irreversibly switched. In total this leads to approximately the same net magnetization of the microstructure due to averaging. In general this behavior is more dominating for the flat models than for the deeper ones.
In addition the second line of the figure shows that the grains are switching in clusters. In detail the upper side of the microstructure reversed the magnetization already while the lower side still has grains with a magnetization anti parallel to the applied external field. This behavior is very common during the reversal process and it is related to the near field interactions of the demagnetization fields. This procedure is well preserved by all autoencoders which is a very important point. However flat_16 reproduces the magnetization states at the stronger external field better than the other models. At this point that model shows less smoothing effects.

In total the autoencoders preserve the individual states of the magnetization and not only the complete magnetization reversal process. However there are some differences to the ground truth related to smoothing effects. This indicates that reconstructed images eventually do not obtain a state where the total energy is minimized. Nonetheless one has to recap that the features of interest are the macroscopic magnetic properties of the microstructure and not the local properties like the switching fields of the individual grains. In that sense slight differences between the ground truth and the reproduced magnetization states can be tolerated as long as

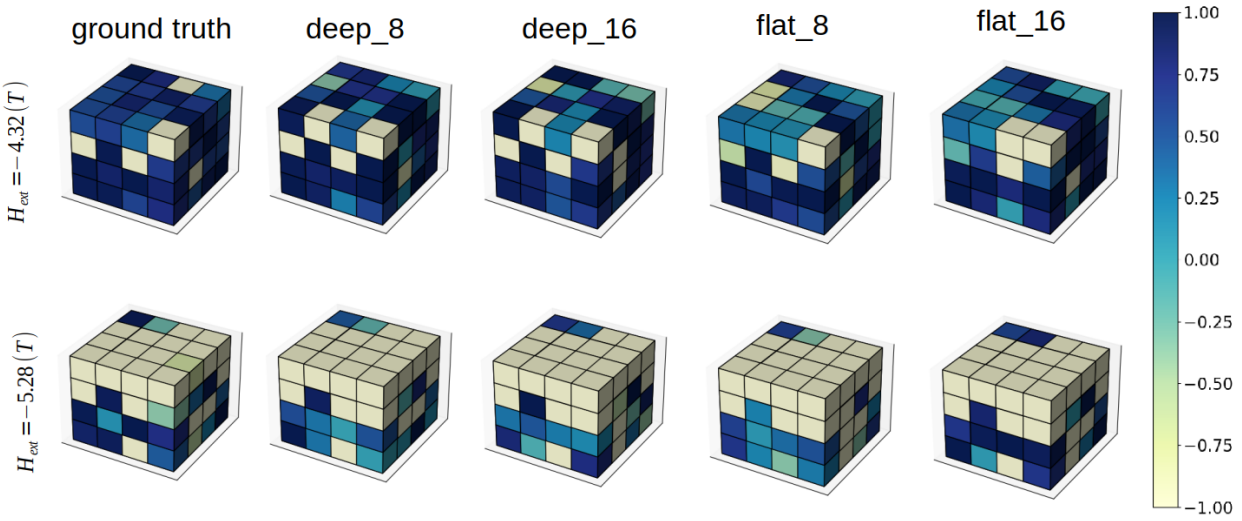

Figure 5.9: Full demagnetization curve of example microstructure. Ground truth labels the results of the reduced order model. Reproduced images are the predictions of the different autoencoders given in section 5.2.2.

the macroscopic properties are estimated sufficiently good.

5.3 Predictor

The autoencoder can be used for dimensionality reduction but the main goal is to predict the complete magnetization reversal process of unseen microstructures. The reason for using machine learning for this task is that these methods are able to reduce the calculation time compared to numerical methods drastically. In general the reduced order model is able to simulate microstructures consisting of thousands of grains within a reasonable amount of time but it is too expensive to optimize the macroscopic magnetic properties of structures at the length scale of applications based on material properties. This section outlines a machine learning model which is able to predict the next state of magnetization in a demagnetization curve by only using a limited amount of previous magnetization states and the external field. Once again this model is implemented for the same simple microstructures as the autoencoder but this should prove the concept.

Before going into the details of the predictor, it is necessary to outline the architecture of the complete algorithm. Therefore figure 5.10 shows a sketch of the method.
The algorithm starts by using the encoder of the autoencoder to obtain embedded images of the magnetization at the i-t, ..., i-1 steps of the demagnetization curve. Furthermore the predictor model takes these images in the latent space and the external field of the i-1-th step of the hysteresis as the input to predict the magnetization in the latent space at the i-th step of the demagnetization curve. Last the decoder must be used to transform the new magnetization state back to the real space. In total this predictor can be used to estimate the complete hysteresis of a magnetic material by only using the first k magnetization states since it can be applied to its own predictions. In total this displays a method where the reduced order model is replaced by a machine learning model and this should lead to a substantial speedup. However the predictor must calculate accurate results that one can use this method.

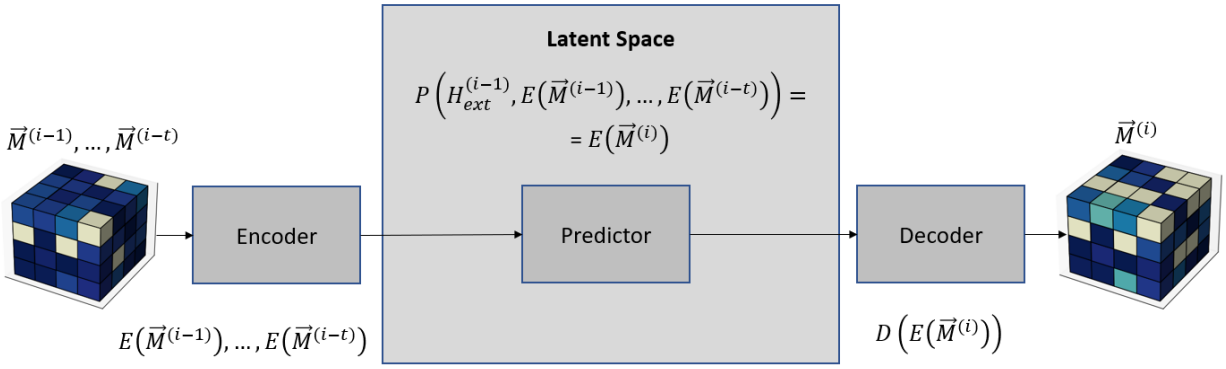

Figure 5.10: Sketch of the method to predict the next step of the hysteresis by using an autoencoder and a predictor in the latent space.

5.3.1 Recursive Training

In general the basic data set to train and test the model is the same as for the autoencoders but it needs to be preprocessed. In detail the first thing to do is to choose a single model for the dimensionality reduction. This is the autoencoder flat_16 since it shows the best performance on the given data as it is analyzed section 5.2. Furthermore the encoder of this model is used to obtain the embedded images of all states of magnetization of the 10 simulations. Again one full simulation is taken as the unseen test set to evaluate the models.
The next step is to concatenate the training samples in batches of t consecutive magnetization states. To avoid misunderstandings, it is worth to note that one magnetization state can be part of at most t different input batches. In addition the strength of the external field at the i-1th step is added to these batches. In total this leads to 1000-t inputs for each simulation.

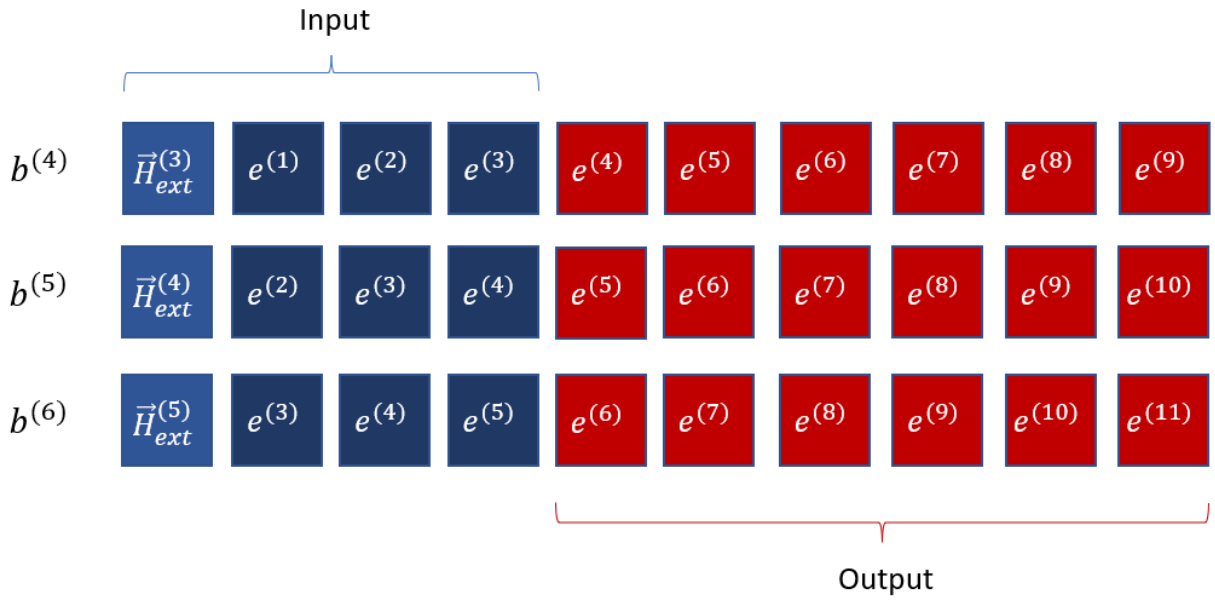

Figure 5.11: Training samples of predictor for t=3 and l=6.

However the next question to answer is how to define a loss function for an empirical risk minimization problem for the model. In general the predictor should be able to estimate the

complete demagnetization curve and not only the next magnetization state. Therefore it must be able to provide accurate results even if the inputs are magnetization states which are calculated by the model itself. This is the reason why a standard mean squared error loss function which only measures the difference to the next state in magnetization eventually fails due to interfering errors. To avoid this problem, one can use a loss function which also accounts for a mean squared error of further predictions. In summary one training sample of the predictor has the following structure:

$$b^{(i)} := (H_{ext}^{(i-1)}, e^{(i-t)}, ...e^{(i)}, e^{(i+1)}, ..., e^{(i+l-1)}) =$$
$$= (H_{ext}^{(i-1)}, E(\vec{M}(H_{ext}^{(i-t)})), ..., E(\vec{M}(H_{ext}^{(i)})), E(\vec{M}(H_{ext}^{(i+1)})), ..., E(\vec{M}(H_{ext}^{(i+l-1)}))),$$

where $e^{(i)} := E(\vec{M}(H_{ext}^{(i)}))$ denotes the encoded magnetization of the i-th external field step. The i-th batch contains the encoded magnetization patterns of the t previous field steps and the l future field steps.
Furthermore the following notation is introduced:

$$\epsilon^{(j)} := \begin{cases} e^{(j)} & \text{if } j < i \\ \hat{e}^{(j)} & \text{else} \end{cases}. \tag{5.3}$$

Here $\hat{e}^{(j)} := P(H_{ext}^{(i)}, \epsilon^{(j-1)}, ..., \epsilon^{(j-t)})$ is the output of the predictor at the j-th step of the demagnetization curve. Using these notations, the mean squared error of one prediction is given by the following formula:

$$L_j(\hat{e}^{(j)}, e^{(j)}) = \sum_{t=1}^{k} (e_t^{(j)} - \hat{e}_t^{(j)})^2, \tag{5.4}$$

where k is the number of latent space variables. Finally it is necessary to sum up the mean squared errors for each prediction to get the loss function:

$$L_{PRED}(\hat{e}, e) = \sum_{j=0}^{l-1} L_j(\hat{e}^{(j)}, e^{(j)}). \tag{5.5}$$

This procedure might look complicated but in detail it is nothing than training the predictor in a way that it can use its own estimates to calculate further magnetization states.
Figure 5.12 outlines the training process of the predictor graphically. The sketch uses a model where t=4 and l=8. In this case the neural network takes the four previous magnetization states as an input. Furthermore it is applied recursively to estimate the next 8 magnetization states and the sum of the means squared error of these predictions is minimized during the training process.

5.3.2 Architecture Optimization

The predictors are operating in the latent space of the autoencoder. Furthermore they are fully connected feedforward neural networks. However the question stays how to define the architecture and the parameters t and l for the model to obtain optimal results. This is again a hyperparameter optimization and therefore it is necessary to compare different setups.

First it is necessary to compare different layer structures for the predictor and therefore t is set to 4 and l is equal to 8 for these experiments. In general the output dimension of the predictor is the number of latent space variables since it should predict only the next magnetization in one iteration. However it is possible to try models with a different number hidden layers. The model with the largest number of hidden layers is given by table 5.3.
The input shape of the predictor models is 65 since it contains four encoded magnetization states

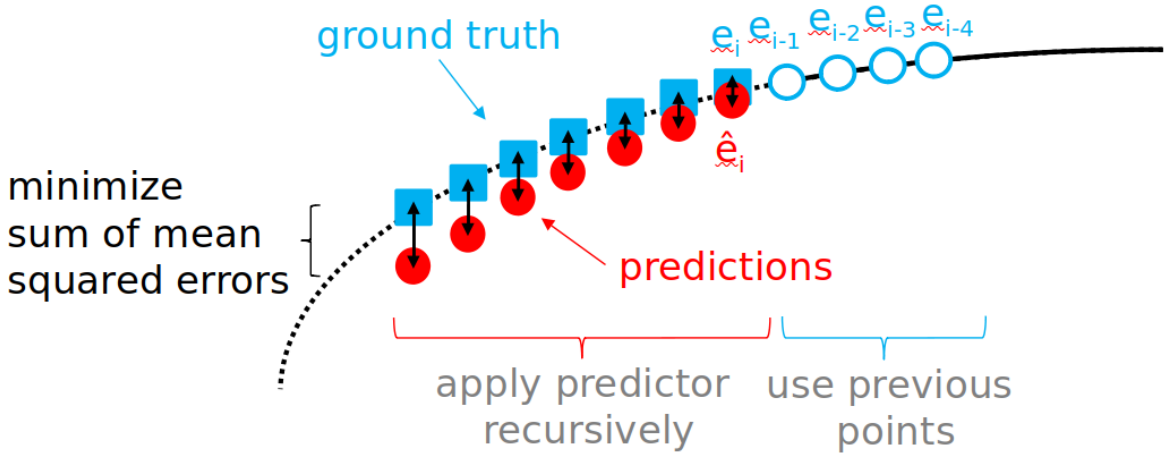

Figure 5.12: Training process of a predictor model with t=4 and l=7.

and the external field. Remember that the magnetization of the microstructure is represented by 16 latent space variables for the autoencoder flat_16. In addition it stacks 6 dense layers where the last one is the output layer. In total this model has 10992 trainable weights. It is further referred as pred_5 in this work.

Layer (type)	Output Shape	Param #
input (InputLayer)	[(None, 65)]	0
dense (Dense)	(None, 64)	4224
dense_1 (Dense)	(None, 64)	4160
dense_2 (Dense)	(None, 32)	2080
dense_3 (Dense)	(None, 32)	1056
dense_4 (Dense)	(None, 16)	528
output (Dense)	(None, 16)	272
Total params: 12,320		
Trainable params: 12,320		
Non-trainable params: 0		

Table 5.3: Model architecture of pred_5.

Another model which is tested in this section is named pred_2. It has the same architecture as pred_5 but with only two hidden layers. Here the first one consists of 64 neurons and the second one has 32 neurons. This predictor has only 6832 trainable parameters. Finally the last neural network under evaluation connects the input layer directly to the output layer. This model is named pred_0. In this case there are only 1056 trainable parameters. In the total the tests with these models should give a brief overview of how deep the predictor must be to provide accurate results.
In addition all layers of the models use the exponential linear unit activation function. Furthermore the weights and biases are again initialized by 'he_normal'.

Moreover they are trained by the same algorithms as the autoencoder where the loss function is given by equation (5.5). In detail it uses backpropagation to update the parameters during the training where the Nadam method is the optimizer. The initial step length of the numerical minimizer is set to 0.0001 and the Keras callback 'ReduceLROnPlateau' is applied to reduce the step length when the loss reaches a plateau during training. Finally early stopping is used to avoid

overfitting. Last the validation set contains 20 % of the training set and it is drawn randomly from the 9000-9t samples.

Learning Curves

The first comparison of the models is done by learning curves. The graphs of interest are plotted in figure 5.13. As previously, they show the loss of the models on the training set respectively the validation set in dependence to the number of epochs.
The learning curves clearly show that the training succeeds and the models are neither underfitting nor overfitting. Furthermore the overall loss of the predictors with hidden layers is clearly smaller than the loss for model without a hidden layer. This indicates that the simple model is not complex enough to approximate the underlying distribution in more detail. This behavior changes for the other models which is eventually related to the fact that they have more degrees of freedoms in the form of trainable parameters.
However the plots of the learning curves have a different shape than the graphs for the autoencoders. In detail the loss is oscillating when using the initial step length for the optimizer. This behavior can be reasoned by the structure of the loss surfaces of the model. This high dimensional surface eventually has local minima nearby each other and a greater step length can cause the optimizer to jump between different extrema [43]. However the 'ReduceLROnPlateau' callback lowers the step length if the validation loss does not reduce significantly after a limited amount epochs. This shrinking of the step length finally enables the optimizer to converge to a deep´ local minimum.

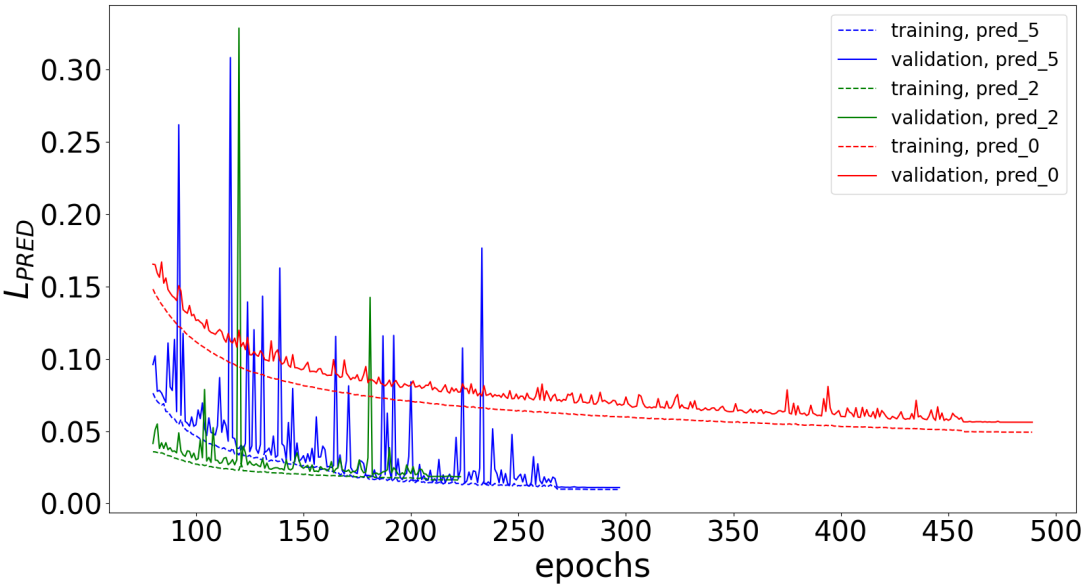

Figure 5.13: Learning curves of different predictor models with t=4 and l=8.

The total loss of the predictor is the sum over the functions given by equation (5.4). However if the predictor should be able calculate a complete hyeresis loop by just having the initial magnetization states, it is necessary that the error of future predictions is limited. To analyze that error, one can compare the contribution of each term to the total loss. Figure 5.14 displays each term of the sum of equation (5.5) separately for the predictor with five hidden layers evaluated on the validation set.
The graphs show that all contributions to the loss are approximately equal after finishing the training. However the further the model predicts the more the contributions oscillate. This indicates that measuring the mean squared error of further predictions introduces more local minima

in the loss surface of the model. Nevertheless the optimizer can find a sufficiently small minima if the step length is reduced during training. Nonetheless it is worth to note that is necessary to reduce the step length during training since starting with a smaller initial step length eventually causes the algorithm to be kept in a local minima which is by far higher than the global minimum [51].

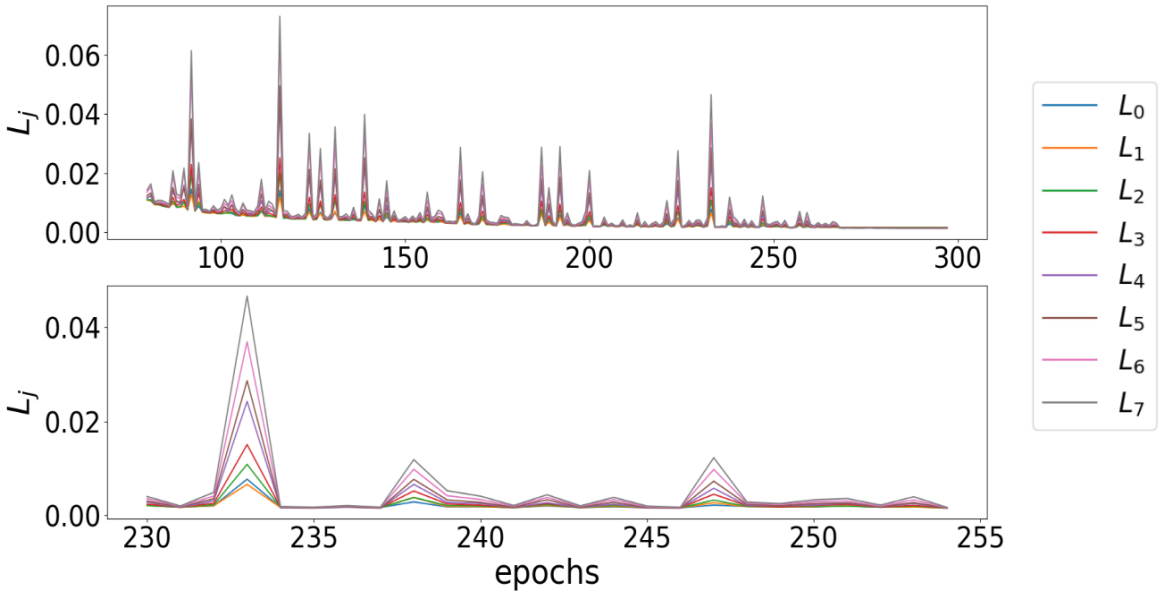

Figure 5.14: Error contributions of pred_5 with t=4 and l=8 evaluated for the validation set.

Predicted Demagnetization Curve

The predictor models are trained in a way that they should be able to predict the complete hysteresis of an unseen microstructure by using just the first few magnetization states. Here we evaluate the performance of the predictor for the unseen demagnetization curve. This means that it is necessary to use the reduced order model to calculate the first input of the model. Afterwards it is possible use the predictor on its own estimations to obtain the complete magnetization reversal process. Figure 5.15 shows the hysteresis of the test data calculated by the reduced order model and the predictions of the models in the real space.
The plots clearly show that the predictor model must be deep enough to able to provide accurate results. In detail one can see that the predictor without a hidden layer can not estimate the hysteresis curve. Furthermore the model with just two intermediate layers produces good results but it is clearly outperformed by the deepest neural network.
However also the best performing machine learning algorithm shows some differences to the ground truth. The predicted hysteresis of pred_5 tends to be more square than the curve of the reduced order model which eventually would lead to a larger $(BH)_{max}$ when trying to estimate this macroscopic magnetic property by machine learning. However the coercivity as well as the remanence of the magnetic object is approximated reasonably well.

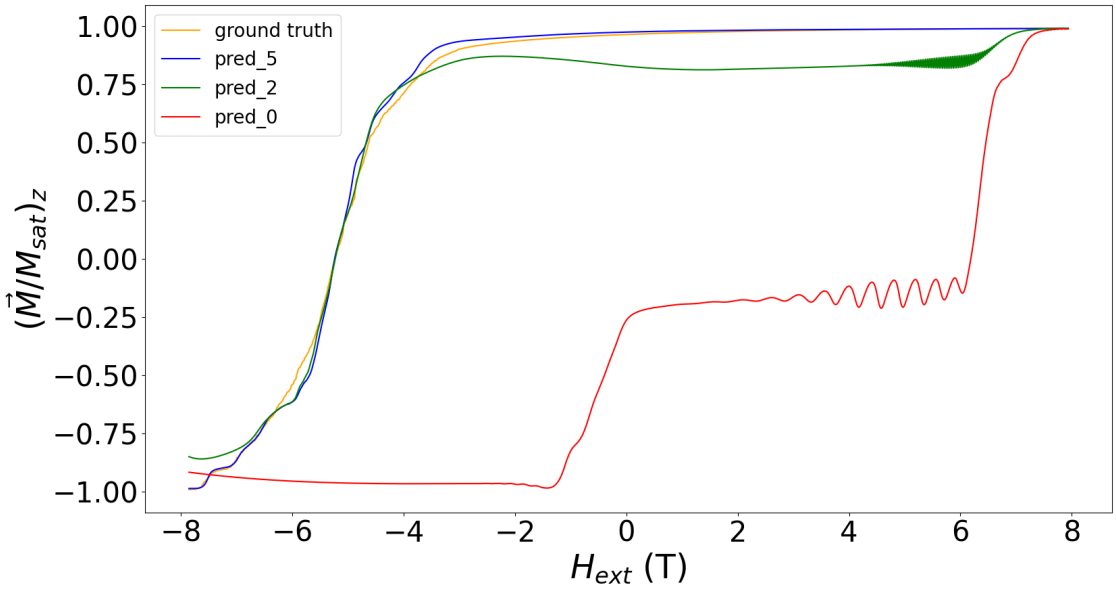

Figure 5.15: Demagnetization curve of microstructure calculated by reduced order model and the predictor models with $t = 4$ and $l = 8$

5.3.3 Hyperparameter Optimization

So far it is evaluated that the model must have enough hidden layers to be able to represent the underlying distribution. However there are more hyperparameters to optimize. In detail the loss function given by equation (5.5) introduces the parameter l which defines how many future predictions are minimized during the training. In addition it is eventually possible to improve the performance of the model by providing more input values from previous field steps which is controlled by the parameter t.

The following analysis of the parameters t and l is done with the model pred_5 which is given by table 5.3. However the input dimension is adapted to the number of previous magnetization states. This model is trained by backpropagation with an Nadam optimizer. Moreover the initial step length is set to 0.0001 and it is reduced if the loss reaches a plateau. Furthermore early stopping is applied to avoid overfitting.

Learning Curves

Figure 5.16 shows the learning curves of the model with different pairs of t and l.
The plots indicate that the model successfully reduces the loss during training for all combinations of t and l. Moreover they do not seem to overfit since the error on the validation set decreases accordingly to the error on the training set until the training is finished. However the graphs clearly show that enlarging either t or l does increase the oscillation when using the initial step length of the optimizer. This behavior reproduces the results from the previous section. Furthermore the absolute loss of the configuration $t = 4$ and $l = 4$ is the smallest which is the expected behavior since increasing l adds additive terms to the loss function. Moreover if l is equal to 8, the total loss does not change if t is increased from 4 to 8. This leads to the assumption that 4 previous magnetization states are enough to accurately predict the next one in this setup.

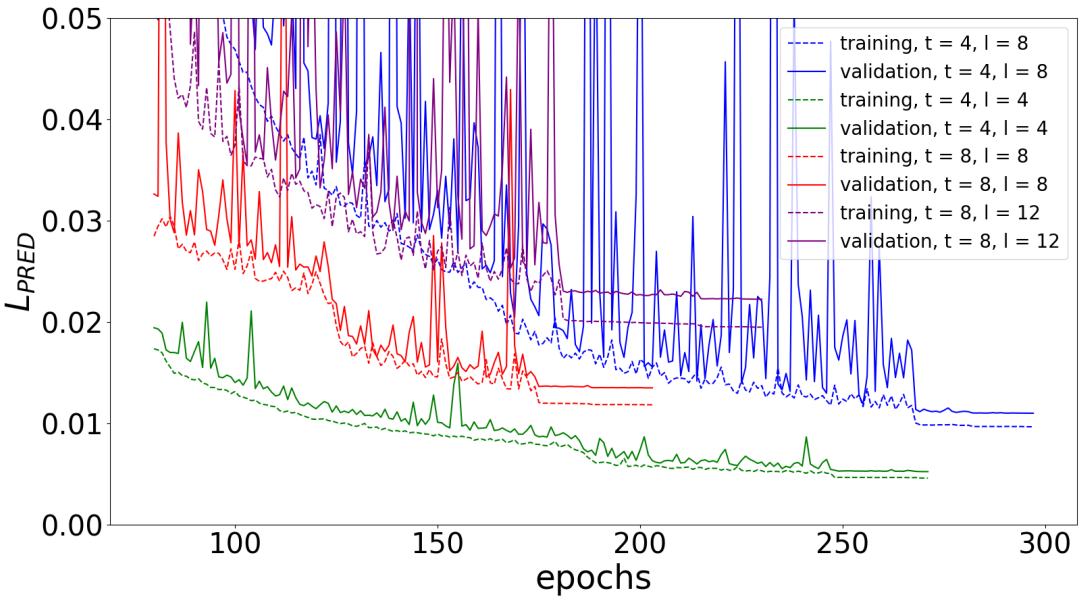

Figure 5.16: Learning curves of pred_5 with different configurations of t and l.

Predicted Demagentization Curve

Finally figure 5.17 shows the complete demagnetization curve of the test microstructure predicted by the model with different configurations of t and l. In addition there is the ground truth which is the calculate hysteresis by the reduced order model.
Although the learning curves of figure 5.16 show that each of the configurations is able to do at least four prediction with sufficiently small error, it is clearly displayed that l must be greater than 4 to enable the model to calculate the complete demagnetization curve reasonably well. In detail the model where $t = 4$ and $l = 4$ does not reproduce the hysteresis and the oscillating predictions are eventually related interfering errors. However if $l = 8$ or greater the model is able to estimate the curve of interest. Nevertheless it is interesting to see that the configuration $t = 8$

and $l = 8$ scores worse than the model with $t = 4$ and $l = 8$. In total the simulations show that the configurations where t is is at least one third less than l outperform the other combinations.

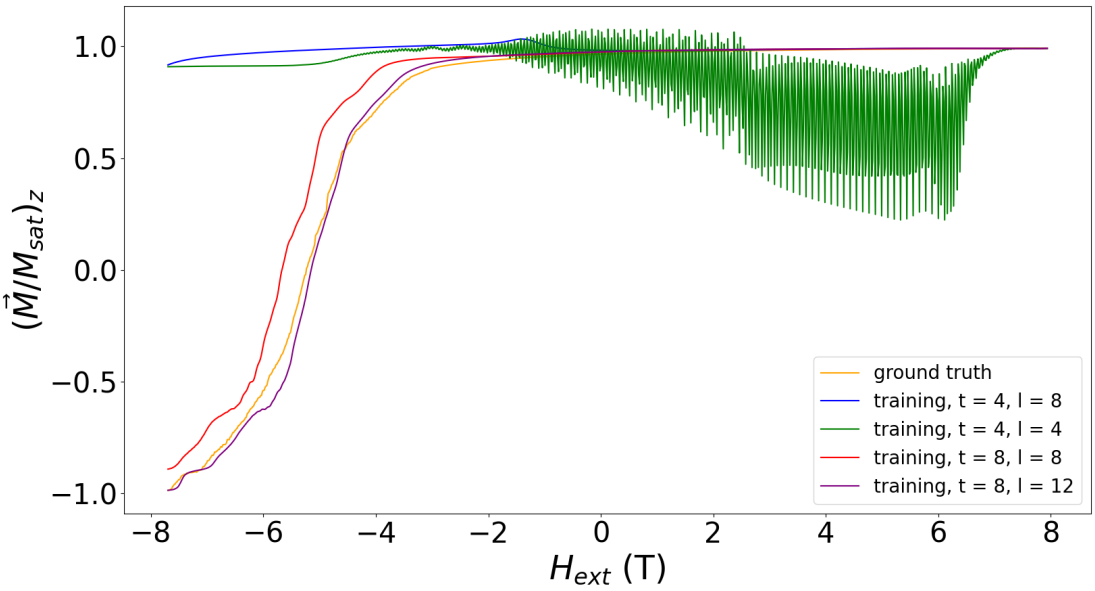

Figure 5.17: Demagnetization curve of microstructure calculated by the reduced order model and predicted by pred_5 with different values of t and l.

In summary the experiments prove that it is possible to predict the hysteresis curve of an unseen microstructure by a machine learning model operating in the latent space of an autoencoder. However there are some important facts which need to be fulfilled to enable the method to succeed. The first is that the model based on neural networks must have enough hidden layers to estimate the underlying distribution. Furthermore it is necessary to recursively apply the predictor during training by using a loss function which takes the mean squared error of future predictions into account. Moreover simply increasing the number of inputs of previous magnetization states does not increase the model performance as long as the number future predictions within the loss function is not changed accordingly.

Chapter 6

Conclusion and Outlook

This thesis first outlined the main properties of high performance magnets and it gave a brief introduction to a common manufacturing process. Furthermore it went into the details of the relation between the macroscopic magnetic properties and the intrinsic magnetic properties respectively the microstructure of the object. Therefore the magnetization reversal process was discussed since this is one of the main tools to compare different magnetic objects and to obtain the figures of interest. Moreover micromagnetism was introduced in brief way where the focus lied on applications with a slowly changing external field. This enables to calculate the demagnetization curve by energy minimization. Furthermore the reduced order model was explained which uses the analytically solvable Stoner-Wohlfarth particle to approximate the magnetic hysteresis of a multigrain microstructure. Although this method is able to provide accurate results, it has the problem that it is computationally not tractable to simulate magnets at an application length scale.
This was the reason to motivate machine learning as the method to sufficiently reduce the computation time to calculate the demagnetization curve of microstructures consisting of thousands of grains. Therefore chapter 3 went from the basics of machine learning towards the very famous neural networks of chapter 4. In detail machine learning was explained as an empirical risk minimization problem coming from statistical learning theory. Moreover the importance of numerical optimizers were discussed on the popular stochastic gradient descent method. Last it introduced techniques like learning curves and k-fold cross validation to successfully apply machine learning in practice.
However chapter 3 built the basis to properly introduce neural networks in 4. In this part of the thesis the backpropagation algorithm with automated differentiation was explained as the method to efficiently train a complex neural network. Furthermore the universal approximation theorem was stated as the motivation for using this type of machine learning model. Last convolutional autoencoders were introduced as a method for dimensionality reduction which is able to conserve nonlinear dependencies within the data set. Therefore it was necessary to explain convolutional layers which are commonly used in image recognition.
Finally the chapter 5 applied the methods explained in the previous section to develop an algorithm to estimate the magnetization reversal process by machine learning. This had required to first calculate a data set by doing simulations with the reduced order model. Furthermore autoencoders with different setups were applied to the data set to reduce the dimensionality of the problem to enable a derivation of a predictor model. Although the experiments must be understood as a proof of concept of this method, we were able to find useful insights which are necessary to take into account if one wants to calculate the demagnetization curve by machine learning.

In total the thesis clearly outlined that loss function engineering is necessary to apply common machine learning techniques to micromagnetism. One insight which was gained during this work is that adding the norm error to the loss function of the autoencoder did provide better results

for this setup than just taking the standard means squared error. In detail adding this term with an appropriate weight led to the same results for the mean squared error while reducing the error in the norm. Moreover the inclusion of this term reduced the training time of the model since it was able to find a minima of the loss functions after less epochs. Furthermore we were able to find differences between the ground truth and the reproduced images which gives room for improvement for this method.
In addition the experiments with the predictors clearly stated the fact that it is necessary to account the error of future prediction during the training of the model to enable it to predict the complete hysteresis curve. Therefore a recursive loss function was developed which respects the mean squared error of the future predictions. Furthermore we were able to show that the neural network for calculating the following steps of the demagnetization curve must be deep enough to approximate the magnetization states accurately. Finally we evaluated architectures for the predictor by trying different combinations of the hyperparameters which describe the number of input magnetization states and the number of recursive iterations during the training. These experiments showed that ratio between these parameters effects the accuracy of the model.

The goal of this thesis was a proof of concept of the machine learning method for calculating the demagnetization curve of 3D microstructures. Nevertheless it is possible to extend this method for synthetic material. However more complex structures eventually need more complex methods. In detail the experiments with the norm error showed that adding information to the autoencoder improves the model. Therefore it would be interesting to apply additional information to the loss function. This can be a term which penalizes the model if the predicted state is not in its equilibrium. This idea comes from the work of [37] which introduces physical informed neural networks where the underlying equations of micromagnetism are part of the loss function. Moreover another promising way to improve the predictor is to enrich its input by information about the microstructure and the intrinsic magnetic properties of the material. However since dimensionality is expected to be a problem one has to find a way to represent the microstructure itself within the latent space.

Deutsche Zusammenfassung

In dieser Arbeit werden die Ummagnetisierungsprozesse von High Performance Permanentmagneten mittels Machine Learning (Maschinelles Learning) analysiert und berechnet.
Hierfür werden zuerst die wichtigen Eigenschaften dieser Permanentmagneten beschrieben. Allgemein werden diese in makroskopische magnetische Eigenschaften und intrinsische magnetische Eigenschaften unterteilt. Im Zuge der makroskopischen Eigenschaften wird das Koerzitivfeld, die Remanenz und das Energie Dichte Produkt erklärt. Diese Eigenschaften sind essenziell für die Verwendung von Permanentmagneten in Gegenständen wie Elektromotoren oder Windrädern. Darüber hinaus sind sie stark von der Mikrostruktur und den intrinsischen Eigenschaften des Magneten abhängig. Aus diesem Grund werden wichtige intrinsische Eigenschaften wie die Curie Temperatur und die magnetokristalline Anisotropie-Achse diskutiert. Darüber hinaus wird ein Herstellungsverfahren für Hot-Deformed High Performance Magneten beschrieben, da die Herstellung maßgeblich zur Mikrostruktur und den intrinsischen Eigenschaften des Magneten beiträgt. Des Weiteren wird die Kontinuumstheorie des Mikromagnetismus eingeführt, da auf dieser theoretischen Basis effiziente Algorithmen zur Berechnung der wichtigen Eigenschaften von Permanentmagneten erstellt werden können. Im Detail wird hier das Reduced Order Model beschrieben, welches eine effiziente Erweiterung des Embedded Stoner Wohlfarth Algorithmus darstellt. Diese Methode kann dazu verwendet werden, um den Ummagnetisierungsprozess von Permanentmagneten zu simulieren. Jedoch ist dieser Algorithmus sehr teuer, wenn Magneten, bestehend aus einer Vielzahl an mikroskopischen magnetischen Körnern, anlysiert werden sollen. Aus diesem Grund wird in der Arbeit erklärt, welche Bedingungen erfüllt sein müssen, damit billigere Machine Learning Algorithmen hier Abhilfe schaffen können.
Dafür wird zuerst Machine Learning mittels statistischer Lerntheorie eingeführt. Im Zuge dessen werden die Basiskomponenten eines Machine Learning Algorithmus diskutiert. In dieser Arbeit ist die Loss-Funktion (Kostenfunktion) von maßgeblicher Bedeutung, da hier der Algorithmus mit physikalischer Theorie angereichert werden kann. Im weiteren Teil der Arbeit liegt der Fokus auf Deep Learning im Kontext von Artificial Neural Networks (Künstliche Neuronale Netzwerke). Im Detail werden Fully-Connected Feedforward Neural Networks beschrieben, da diese später zur Berechnung der Hysteresekurve verwendet werden. Darüber hinaus wird auf Convolutional Neural Networks eingegangen, welche die Basis für Convolutional Autoencoders bilden. Das ist eine Machine Learning Methode, welche verwendet werden kann, um die Dimension des zugrunde liegenden Problems zu reduzieren. Das ist notwendig, da Vorhersagen der Ummagnetisierungskurve von komplexen Magneten im realen Raum nicht möglich sind, da hierfür die notwendige Rechenleistung nicht zur Verfügung steht. In anderen Worten, hätten Prädiktoren im realen Raum zu viele Gewichte, um sie effizient trainieren zu können. Im Allgemeinen können Feedforward Neural Networks mittels des Backpropagation Algorithmus effizient trainiert werden, auf welchen in der Arbeit näher eingegangen wird.
Im letzten Teil der Arbeit wird Machine Learning dazu verwendet, um die Hysteresekurve von einfachen Mikrostrukturen zu bestimmen. Hierfür wird zuerst ein Autoencoder trainiert. Dieser verwendet eine Loss-Function, welche zusätzlich zum Mittleren Quadratischen Fehler (MSE) auch die Normerhaltung berücksichtigt. Im Detail wird die Loss-Function des MSE um einen additiven Term erweitert, welcher diese grundlegende mikromagnetische Regel miteinbezieht. Darüber hinaus wird ein Prädiktor-Modell im Latent Space des Autoencoders beschrieben und analysiert. Dieses Modell nimmt Magnetisierungszustände von vorhergehenden Punkten der Um-

magnetisierungskurve und die Stärke des externen Feldes, um die nächsten Schritte der Hysterese vorherzusagen. Auch hier liegt der Fokus auf der Loss-Function, da diese garantieren muss, dass der Prädiktor mehr als einen Schritt der Ummagnetisierungskurve berechnen kann. Im Detail muss hierfür das Modell während des Trainings auf seine eigenen Vorhersagen angewendet werden. Die gesamte Loss-Function bezieht Fehler der zukünftigen Berechnungen mit ein. Das ist notwendig, da das grundlegende Ziel ist, komplette Hysteresekurven mittels Machine Learning zu berechnen.

Zusammenfassend zeigen die numerischen Experimente der Arbeit, dass eine repräsentative Loss-Function für beide Modelle notwendig ist, damit der Machine Learning Algorithmus zufriedenstellende Ergebnisse liefert. Die Berechnungen ergeben, dass der Term der Normerhaltung, in der Loss-Function des Autoencoders, die Trainingszeit reduziert, die Genauigkeit bezüglich der Normen der reproduzierten Bilder erhöht und darüber hinaus äquivalente Werte für den MSE liefert, verglichen mit einer Loss-Function, welche nur den MSE respektiert. Nichtsdestotrotz ist es wichtig zu erwähnen, dass die Gewichtung der beiden Terme gut gewählt sein muss, da das Modell sonst ungenaue Ergebnisse liefert. Darüber hinaus zeigen Vergleiche zwischen den originalen Magnetisierungszuständen und den reproduzierten Magnetisierungszuständen leichte Unterschiede. Diese Unterschiede sind im Zuge dieser Arbeit auf Schmiereffekte des Autoencoders zurückgeführt. Interessant ist, dass diese Unterschiede die vorhergesagte Hysterese weniger beeinflusst, als die Visulisierungen deutlich machen würden.
Des Weiteren zeigt die Analyse des Prädiktor-Modells, dass es notwendig ist, das Modell während des Trainings rekursiv anzuwenden. Ohne des rekursiven Trainings, ist das Modell nicht im Stande ganze Hysteresekurven akkurat zu berechnen. Darüber hinaus lassen die Ergebnisse darauf schließen, dass das Verhältnis zwischen rekursiven Anwendungen und Input Magnetisierungszustände maßgeblich zur Genauigkeit des Modells beitragen. Außerdem muss das Modell ein verhältnismäßig tiefes Neural Network sein, damit die zugrundeliegende Verteilung gut abgebildet werden kann.

Die Grundidee dieser Arbeit ist zu überprüfen, ob eine Machine Learning Methode bestehend aus Autoencoder und Prädiktor im Latent Space die Ummagnetisierungskurve von Mikrostrukturen vorhersagen kann. Für diesen Proof of Concept werden einfache Strukturen verwendet. Im Rahmen dieser Arbeit unterscheiden sich diese Strukturen in der magnetokristallinen Anisotropie-Achse der zugrunde liegenden magnetischen Körnern. Das bedeutet, dass das trainierte Modell in der Lage ist, die Hysteresekurve von Mikrostrukturen vorherzusagen, welche eine komplett neue Simulation durch das Reduced Order Model benötigen würden. Der Vorteil hier ist, dass die Evaluation des Machine Learning Modells kostengünstiger ist, als die Berechnung des numerischen Algorithmus.
Obwohl im Zuge der Arbeit nur einfache Strukturen verwendet werden, kann diese Methode auf synthetische Mikrostrukturen erweitert werden. Das erfordert jedoch eine Anpassung der Modelle. Im Detail zeigen die Experimente, dass physikalische Eigenschaften, beschrieben durch die Loss-Function, die Algorithmen akkurater machen. Aus diesem Grund gibt es Raum für Verbesserung, indem zusätzliche physikalische Informationen, abseits der Normerhaltung, in der Loss-Function abgebildet werden. Ein Beispiel dafür wäre, die Loss-Function mit einem additiven Term anzureichern, welcher die Energiegleichung des statischen Mikromagnetismus beinhaltet. Diese Idee nennt sich Physics Informed Neural Networks und ist eine beliebte Technik in der derzeitigen Forschung. Ein anderer Weg zur Verbesserung des Prädiktors ist es, dem Modell zusätzliche Inputs über die Mikrostruktur zu übergeben. Hierfür müsste ein neuer Autoencoder trainiert werden, welcher es ermöglicht, die Mikrostruktur in einem niederdimensionalen Raum darzustellen, da sonst erneut die hohe Dimensionalität ein Problem darstellen könnte.

Zusammenfassend beschreibt diese Arbeit eine Machine Learning Methode, welche zur Berech-

nung der Ummagnetisierungskurve eines Permanentmagneten verwendet werden kann. Die Analysen zeigen klar, dass Machine Learning effizient und mit hoher Genauigkeit diesen sehr wichtigen physikalischen Prozess berechnen kann.

Bibliography

[1] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] A. F. Agarap. Deep Learning using Rectified Linear Units (ReLU). *arXiv:1803.08375*, 2018.

[3] N. Ashcroft, A. W, N. Mermin, W. Ashcroft, D. Mermin, N. Mermin, and B. P. Company. *Solid State Physics*. HRW international editions. Holt, Rinehart and Winston, 1976.

[4] C. R. H. Bahl. Estimating the demagnetization factors for regular permanent magnet pieces. *AIP Adv.*, 11(7):075028, July 2021.

[5] A. Barron. Universal approximation bounds for superpositions of a sigmoidal function. *IEEE Trans. Inform. Theory*, 39(3):930–945, May 1993.

[6] J. O. Berger and C. Srinivasan. Generalized Bayes estimators in multivariate problems. *Ann. Statist.*, 6(4):783–801, July 1978.

[7] J. Berner, P. Grohs, G. Kutyniok, and P. Petersen. The modern mathematics of deep learning. *arXiv:2105.04026*, 2021.

[8] D. Berrar. Cross-validation. In *Encyclopedia of Bioinformatics and Computational Biology*, pages 542–545. Elsevier, Jan. 2019.

[9] K. Binnemans, P. T. Jones, T. Müller, and L. Yurramendi. Rare earths and the balance problem: How to deal with changing markets? *J. Sustain. Metall.*, 4(1):126–146, Feb. 2018.

[10] C. J. Brown, J. R. Mullaney, J. Morrison, J. W. Martin, and T. J. Trombley. Chloride concentrations, loads, and yields in four watersheds along interstate 95, southeastern connecticut, 2008-11: Factors that affect peak chloride concentrations during winter storms. *USGS Publications Warehouse*, 2015.

[11] W. F. Brown. Micromagnetics, domains, and resonance. *J. Appl. Phys.*, 30(4):S62–S69, Apr. 1959.

[12] N. Buduma. *Fundamentals of Deep LearningDesigning Next-Generation MachineIntelligence Algorithms*. O'Reilly Media, 1st edition, 2017.

[13] S. Börm. Matrix arithmetics in linear complexity. *Computing*, 77(1):1–28, Dec. 2005.

[14] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (ELUs). *arXiv:1511.07289*, 2016.

[15] J. Coey. Perspective and prospects for rare earth permanent magnets. *Engineering-london.*, 6(2):119–131, Feb. 2020.

[16] F. Cucker and S. Smale. On the mathematical foundations of learning. *Bull. Amer. Math. Soc.*, 39(1):1–49, Oct. 2001.

[17] L.-F. Dong, Y.-Z. Gan, X.-L. Mao, Y.-B. Yang, and C. Shen. Learning deep representations using convolutional auto-encoders with symmetric skip connections. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, editors, *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, volume 29. IEEE, Apr. 2018.

[18] T. Dozat. Incorporating Nesterov momentum into Adam. *(n.p.)*, 2016.

[19] L. Exl, J. Fischbacher, A. Kovacs, H. Özelt, M. Gusenbauer, K. Yokota, T. Shoji, G. Hrkac, and T. Schrefl. Magnetic microstructure machine learning analysis. *J. Phys. Mater.*, Nov. 2018.

[20] L. Exl, D. Suess, and T. Schrefl. Micromagnetism. In M. Coey and S. Parkin, editors, *Handbook of Magnetism and Magnetic Materials*, pages 1–44. Springer International Publishing, Cham, 2021.

[21] J. Fischbacher, A. Kovacs, L. Exl, J. Kühnel, E. Mehofer, H. Sepehri-Amin, T. Ohkubo, K. Hono, and T. Schrefl. Searching the weakest link: Demagnetizing fields and magnetization reversal in permanent magnets. *Scripta Mater.*, 154:253–258, Sept. 2018.

[22] J. Fischbacher, A. Kovacs, M. Gusenbauer, H. Oezelt, L. Exl, S. Bance, and T. Schrefl. Micromagnetics of rare-earth efficient permanent magnets. *J. Phys. D: Appl. Phys.*, 51(19):193002, Apr. 2018.

[23] K. Friston and W. Penny. Empirical Bayes and hierarchical models. In *Statistical Parametric Mapping*, pages 275–294. Elsevier, London, 2007.

[24] Z. Gan and Z. Xu. Efficient implementation of the Barnes-hut octree algorithm for Monte Carlo simulations of charged systems. *Sci. China Math.*, 57(7):1331–1340, Feb. 2014.

[25] A. Gron. *Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems*. O'Reilly Media, Inc., 1st edition, 2017.

[26] D. Guptasarma and B. Singh. New scheme for computing the magnetic field resulting from a uniformly magnetized arbitrary polyhedron. *Geophysics*, 64(1):70–74, Jan. 1999.

[27] O. Gutfleisch, M. A. Willard, E. Brück, C. H. Chen, S. G. Sankar, and J. P. Liu. Magnetic materials and devices for the 21st century: Stronger, lighter, and more energy efficient. *Adv. Mater.*, 23(7):821–842, Dec. 2010.

[28] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on ImageNet classification. *IEEE International Conference on Computer Vision (ICCV 2015)*, 1502, Feb. 2015.

[29] W. Heisenberg. Zur Theorie des Ferromagnetismus. *Z. Physik*, 49(9-10):619–636, Sept. 1928.

[30] K. Hioki. High performance hot-deformed Nd-Fe-b magnets (Review). *Sci. Technol. Adv. Mat.*, 22(1):72–84, Jan. 2021.

[31] M. Iwasawa, D. Namekata, R. Sakamoto, T. Nakamura, Y. Kimura, K. Nitadori, L. Wang, M. Tsubouchi, J. Makino, Z. Liu, H. Fu, and G. Yang. Implementation and performance of Barnes-hut n-body algorithm on extreme-scale heterogeneous many-core architectures. *The International Journal of High Performance Computing Applications*, 34(6):615–628, July 2020.

[32] I. T. Jolliffe and J. Cadima. Principal component analysis: A review and recent developments. *Phil. Trans. R. Soc. A.*, 374(2065):20150202, Apr. 2016.

[33] J. Kalezhi. *Modelling data storage in nano-island magnetic materials*. PhD thesis, University of Manchester, Faculty of Engineering and Physical Sciences, 2011.

[34] T. Kim, J. Oh, N. Kim, S. Cho, and S.-Y. Yun. Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation. *arXiv:2105.08919*, 2021.

[35] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv:1412.6980*, 2017.

[36] A. Kovacs. *Novel numerical studies of bit patterned magnetic recording*. PhD thesis, Technische Universität Wien, 2020.

[37] A. Kovacs, L. Exl, A. Kornell, J. Fischbacher, M. Hovorka, M. Gusenbauer, L. Breth, H. Oezelt, M. Yano, N. Sakuma, A. Kinoshita, T. Shoji, A. Kato, and T. Schrefl. Conditional physics informed neural networks. *Commun. Nonlinear Sci.*, 104:106041, Jan. 2022.

[38] A. Kovacs, J. Fischbacher, H. Oezelt, M. Gusenbauer, L. Exl, F. Bruckner, D. Suess, and T. Schrefl. Learning magnetization dynamics. *J. Magn. Magn. Mater.*, 491:165548, Dec. 2019.

[39] H. Kronmüller and M. Fähnle. *Micromagnetism and the Microstructure of Ferromagnetic Solids*. Cambridge University Press, 1st edition, 2009.

[40] H. Kronmüller. Theory of nucleation fields in inhomogeneous ferromagnets. *phys. stat. sol. (b)*, 144(1):385–396, Nov. 1987.

[41] H. Kronmüller, K.-D. Durst, and G. Martinek. Angular dependence of the coercive field in sintered Fe77Nd15B8 magnets. *J. Magn. Magn. Mater.*, 69(2):149–157, Oct. 1987.

[42] S. K. Kumar. On weight initialization in deep neural networks. *arXiv:1704.08863*, 2017.

[43] H. Li, Z. Xu, G. Taylor, and T. Goldstein. Visualizing the loss landscape of neural nets. In *NeurIPS*, 2018.

[44] Z. Lu, H. Pu, F. Wang, Z. Hu, and L. Wang. The expressive power of neural networks: A view from the width. *arXiv:1709.02540*, 2017.

[45] O. Manzyuk, B. A. Pearlmutter, A. A. Radul, D. R. Rush, and J. M. Siskind. Perturbation confusion in forward automatic differentiation of higher-order functions. *J. Funct. Prog.*, 29, 2019.

[46] F. Meszaros, M. Shatanawi, and G. A. Ogunkunbi. Challenges of the electric vehicle markets in emerging economies. *Period. Polytech. Transp. Eng.*, 49(1):93–101, Feb. 2020.

[47] A. Y. Mikhalev and I. V. Oseledets. Iterative representing set selection for nested cross approximation. *Numer. Linear Algebra Appl.*, 23(2):230–248, Nov. 2015.

[48] R. Morris. D.O. hebb: The organization of behavior, wiley: New york; 1949. *Brain Res. Bull.*, 50(5-6):437, Nov. 1999.

[49] G. S. Mueller, Andreas C. *Introduction to machine learning with Python : A guide for data scientists*. O'Reilly Media, 1st edition, 2016.

[50] J. Nagi, F. Ducatelle, G. Di Caro, D. Ciresan, U. Meier, A. Giusti, F. Nagi, J. Schmidhuber, and L. M. Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. *2011 IEEE International Conference on Signal and Image Processing Applications, ICSIPA 2011*, pages 342–347, Nov. 2011.

[51] K. Nar and S. Sastry. Step size matters in deep learning. *arXiv:1805.08890*, May 2018.

[52] S. Ntanos, M. Skordoulis, G. Kyriakopoulos, G. Arabatzis, M. Chalikias, S. Galatsidas, A. Batzios, and A. Katsarou. Renewable energy and economic growth: Evidence from European countries. *Sustainability*, 10(8):2626, July 2018.

[53] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall. Activation functions: Comparison of trends in practice and research for deep learning. *arXiv:1811.03378*, Dec. 2020.

[54] K.-S. Oh and K. Jung. GPU implementation of neural networks. *Pattern Recogn.*, 37(6):1311–1314, June 2004.

[55] J. Parkinson and D. J. J. Farnell. *An Introduction to Quantum Spin Systems*, volume 816. Springer Berlin Heidelberg, 2010.

[56] V. Pestov. PAC learnability versus VC dimension: A footnote to a basic result of statistical learning. *arXiv:1104.2097*, 2011.

[57] A. Radhakrishnan, K. D. Yang, M. Belkin, and C. Uhler. Memorization in overparameterized autoencoders. *arXiv: Computer Vision and Pattern Recognition*, 2018.

[58] C. Rey and A. Malozemoff. Fundamentals of superconductivity. In *Superconductors in the Power Grid*, pages 29–73. Elsevier, 2015.

[59] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv:1609.04747*, 2017.

[60] M. Sato and Y. Ishii. Simple and approximate expressions of demagnetizing factors of uniformly magnetized rectangular rod and cylinder. *J. Appl. Phys.*, 66(2):983–985, July 1989.

[61] S. Schaffer, N. Mauser, T. Schrefl, D. Suess, and L. Exl. Machine learning methods for the prediction of micromagnetic magnetization dynamics. Master's thesis, Universität Wien, Apr. 2021.

[62] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan, and D. Hassabis. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv:1712.01815*, 2017.

[63] G. V. Skrotskiĭ. The Landau-Lifshitz equation revisited. *Sov. Phys. Usp.*, 27(12):977–979, Dec. 1984.

[64] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer New York, 1995.

[65] D. Wolpert and W. Macready. No free lunch theorems for optimization. *IEEE Trans. Evol. Computat.*, 1(1):67–82, Apr. 1997.

[66] J. Wu. Introduction to convolutional neural networks. *(n.p.)*, 2017.

[67] Y. Yang, A. Walton, R. Sheridan, K. Güth, R. Gauß, O. Gutfleisch, M. Buchert, B.-M. Steenari, T. Van Gerven, P. T. Jones, and K. Binnemans. REE recovery from end-of-life NdFeB permanent magnet scrap: A critical review. *J. Sustain. Metall.*, 3(1):122–149, Sept. 2016.

[68] J.-G. Yoo, H.-R. Cha, T.-H. Kim, D.-H. Kim, Y.-D. Kim, and J.-G. Lee. Coercivity improvement in Nd–Cu infiltrated Nd-Fe-b hot-deformed magnets by controlling microstructure of initial HDDR powders. *Journal of Materials Research and Technology*, 14:340–347, Sept. 2021.

[69] Y. Zhang. A better autoencoder for image: Convolutional autoencoder. *(n.p.)*, 2018.

[70] G. Zhao, X. Zhang, and F. Morvan. Theory for the coercivity and its mechanisms in nanostructured permanent magnetic materials. *Rev Nanosci Nanotech*, 4(1):1–25, Apr. 2015.

[71] D.-X. Zhou. Universality of deep convolutional neural networks. *Appl. Comput. Harmon. A.*, 48(2):787–794, Mar. 2020.

[72] H. Özelt. *Exchange coupled ferri-/ferromagnetic composite nanomagnets*. PhD thesis, Technische Universität Wien, 2018.