



MASTERARBEIT | MASTER'S THESIS

Titel | Title

Property prediction of granular microstructures via graph neural networks

verfasst von | submitted by

Felix Lasthofer BSc

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of
Master of Science (MSc)

Wien | Vienna, 2025

Studienkennzahl lt. Studienblatt | Degree
programme code as it appears on the
student record sheet:

UA 066 910

Studienrichtung lt. Studienblatt | Degree
programme as it appears on the student
record sheet:

Masterstudium Computational Science

Betreut von | Supervisor:

Dipl.-Ing. Dr.techn. Lukas Exl Privatdoz.

Acknowledgements

The financial support by the Austrian Federal Ministry for Digitalization and Economic Development, the Christian Doppler Research Association and the European Union (101129888 - MaMMoS, 101135546 - GREENE) is gratefully acknowledged. Further, I acknowledge the support of the University for Continuing Education Krems for providing me with a professional working environment and working materials. I gratefully acknowledge the support and feedback of my supervisor Dipl.-Ing. Dr.techn. Lukas Exl, Privatdoz. Special thanks go to Univ.-Doz. Dipl.-Ing.Dr. Thomas Schrefl who was a great help to me with his advice, experience and ideas. Further, I want to thank Dr. Alexander Kovacs, MSc BSc, Ing. Dr. Harald Özelt for putting up with the many questions I had and many more for being a great team and supporting me with their knowledge. Finally, I thank my parents Elisabeth Lasthofer, Stefan Lasthofer and my girlfriend Nadine Statzberger.

Abstract

The current climate crisis beckons us to change the way we live, especially in the energy and transportation sector. The change to renewable energies and battery electric vehicles would spike the demand of rare earth metals such as neodymium or dysprosium for efficient and strong permanent magnets, which are used in motors and generators. Current research efforts aim to reduce the required amount of rare-earth in permanent magnets, while trying to maintain excellent permanent magnetic properties.

This thesis evaluates the usability of graph neural networks for predicting the properties of magnets, with the aim of supplementing micromagnetic simulations, which are essential for the research of magnets. Graph neural networks are expected to perform well on this problem as they intrinsically use the granular structure of the magnets to predict desired properties. In addition to the chemical composition, the structure is crucial for the magnet's coercive field. A graph neural network for classifying magnets based on their coercivity was developed. The model was tested for four different datasets in varying size as well as features compositions, including physical and geometrical properties. Accuracy was used to establish a measure of performance of the graph neural network. Recall was applied to emphasize the models' performance in classifying minority classes.

We found that even slight variation in the dataset composition of graphs is sufficient for graph neural networks to yield good results with an accuracy above 90%. We can conclude that graph neural networks are a suitable way to predict the properties of magnets making them a valuable tool for supplementing micromagnetic simulations. For future work, we suggest exploring regression models to predict continuous properties such as coercivity and energy product. A further research is needed to evaluate the generalisation capabilities of graph neural networks on grain structures larger than those seen during training.

Kurzfassung

Die derzeitige Klimakrise zwingt uns unsere Lebensweise zu ändern, insbesondere im Energie- und Verkehrssektor. Die Umstellung auf erneuerbare Energien und Elektrofahrzeuge würde die Nachfrage nach seltenen Schwermetallen wie Neodym oder Dysprosium für effiziente und starke Magnete in die Höhe treiben und die Forschung über die Zusammensetzung von Magneten anregen.

In dieser Arbeit wird die Verwendbarkeit von graph neural networks für die Vorhersage der Eigenschaften von Magneten bewertet, um mikromagnetische Simulationen zu beschleunigen, die für die Erforschung von Magneten unerlässlich sind. Es wird erwartet, dass graph neural networks bei diesem Problem gut arbeiten, da sie die Struktur von Materialien nutzen, um gewünschte Eigenschaften vorherzusagen. Zu diesem Zweck wurden vier verschiedene Datensätze mit unterschiedlicher Größe und Merkmalszusammensetzung, einschließlich physikalischer und geometrischer Eigenschaften, trainiert, um zu sehen, wie sie sich auf die Leistung der graph neural networks auswirken. Da es sich bei der Aufgabe um ein Klassifizierungsproblem handelte, werden die Werte für die accuracy und recall als Maßstab verwendet, um die Gesamtleistung und die Leistung des Modells bei der Klassifizierung von Minderheitsklassen zu bewerten.

Wir haben herausgefunden, dass selbst geringe Unterschiede in der Zusammensetzung der Graphen ausreichen, damit graph neural networks gute Ergebnisse mit einer accuracy von über 90% liefern. Wir können schlussfolgern, dass graph neural networks ein geeignetes Mittel sind, um die Eigenschaften von Magneten vorherzusagen, was sie zu einem wertvollen Werkzeug für die Beschleunigung mikromagnetischer Simulationen macht. Für künftige Arbeiten schlagen wir vor, Regressionsmodelle zur Vorhersage kontinuierlicher Eigenschaften zu erforschen und die generalisierung für Magnete zu bewerten, die größer sind als die beim Training beobachteten.

Contents

Acknowledgements	i
Abstract	iii
Kurzfassung	v
1 Introduction	1
1.1 Climate change and climate policy goals	1
2 Magnets	5
2.1 Making magnets	5
2.2 Properties of magnets	5
2.2.1 Hysteresis and coercivity	6
2.2.2 Magnetocrystalline anisotropy and easy-axis	6
2.2.3 Demagnetising energy	8
2.2.4 Exchange constant and magnetic polarisation	8
2.2.5 Zeeman energy (external field energy)	8
2.3 Models and simulations	9
2.3.1 Stoner-Wohlfarth model	9
2.3.2 Reduced order model for permanent magnets	10
3 Graphs and neural networks	13
3.1 Graphs	13
3.1.1 Usage of graphs	13
3.2 Neural networks	16
3.2.1 Architecture	16
3.2.2 Classification or regression task	18
3.2.3 Training	18
3.2.4 Benchmarking	23
3.2.5 Calculations on graphs	25
4 Data and graph representation	29
4.1 Data Sources and analysis	29
4.1.1 Analysing of features	30
4.2 Preprocessing	33
4.3 From data to graphs	33

5 Training and optimization	35
5.1 Neural network architecture	35
5.2 Dataset 1: Low coercive grain	35
5.2.1 Dataset description	36
5.2.2 Feature analysis	36
5.2.3 Training and optimization	38
5.3 Dataset 2: Grain size	43
5.3.1 Dataset description	43
5.3.2 Feature analysis	44
5.3.3 Training and optimization	44
5.4 Dataset 3: Secondary phases	46
5.4.1 Dataset description	46
5.4.2 Feature analysis	47
5.4.3 Training and optimization	48
5.5 Dataset 4: Generalisation	49
5.5.1 Dataset description	49
5.5.2 Feature analysis	50
5.5.3 Training and optimization	53
5.6 Conclusion and outlook	56
 Bibliography	 59

1 Introduction

1.1 Climate change and climate policy goals

One of the biggest if not the biggest challenge for humanity is the impending climate crisis and its consequences. To challenge it the Paris Climate Agreement was brought into existence and signed by 196 Parties at the UN Climate Change Conference. Naturally this agreement sparked research in the direction on how to uphold it [Tes19] and which raw materials will be needed in greater quantities [BCH⁺20]. In accordance with the latter we can partition our resources in three different categories namely renewables, e-mobility and defence. The transportation sector, powered largely by combustion engines, is a major driver of climate change due to its high CO₂ emissions. To bring a proper change to this sector we have to rethink our solutions regarding public and private transport. A change from combustion engines to electric motors is required. The transition to electric engines only makes sense if the used electricity is also from renewable sources. In Figure 1.1 we see two predictions for the climate goals for an average increase of temperature by 2°C and 1.5°C and how these goals can be achieved by exchanging the current fossil fuel motors (depicted in dark gray) with either hybrid (light gray), electric (blue) or hydrogen (green) cars. Especially battery electric vehicles are expected to take over the majority of the market share earliest by 2035 and latest by 2040. Therefore we can argue that there will be an increase in demand for electric motors and for the materials they are made of. Unfortunately the most efficient electric motors are highly depended on strong permanent magnets and these strong permanent magnets are built from rare earth metals such as Neodymium, Dysprosium and Praseodymium. So even though e-mobility can reduce CO₂ emissions, if electricity is sourced from renewable energies, we might run into a supply issue regarding rare earth metals.

So how do we find solutions to quench the increasing demand for electricity in e-mobility preferably without increasing CO₂ emissions. There are multiple ways to tackle these problems such as photovoltaic, hydroelectric and wind energy. Wind energy especially gets highlighted in [BCH⁺20] as a very cost-effective renewable energy technology that will help us in climate change mitigation. However wind energy, as well as hydroelectric energy, depends on generators which in turn need magnets and rare earth metals to work most efficiently. In Figure 1.2 we can see the expected increase of demand for materials used in wind turbines for the years 2030 and 2050. The increase is given in times its current values and three scenarios are visible through the colourisation of the bars. The two light coloured ones are for the low and high demand scenarios and the fully coloured one is for the medium demand scenario. We can easily see that the demand for the different metals increases rapidly and may lead to supply chain issues in the future.

1 Introduction

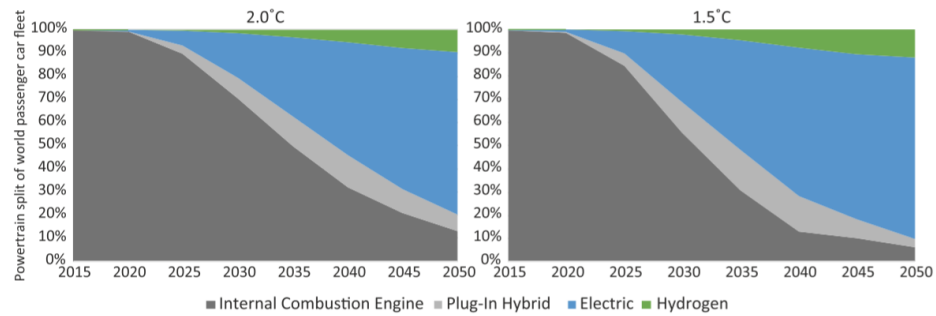


Figure 1.1: Prediction of the division of different engine types depending on climate goals from [Tes19](#)

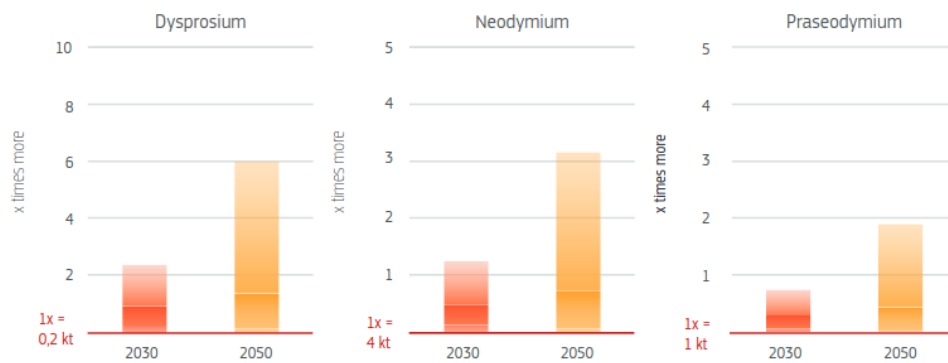


Figure 1.2: Prediction of the increased demand of rare earth metals for wind power. Different predictions can be seen through the faded parts of the bars namely light, medium and high demand scenarios. [BCH⁺20](#)

1.1 Climate change and climate policy goals

We found a solution towards a carbon free economy regarding power generation and transport but stumbled upon the new issue of supply chains regarding rare earth metals. To counter this we can search for different compositions of magnets that either optimize current known microstructures or partially substitute rare earth metals with less critical ones [FKG⁺18, KFO⁺23]. The properties of a permanent magnet depend on the intrinsic magnetic properties and their granular microstructure. The intrinsic magnetic properties such as magnetisation and magnetocrystalline anisotropy arise from the specific crystal structure and associated chemical composition. High performance magnets show a polyhedral grain structure. Grain size, grain boundary phases, and secondary phases such as oxides have a severe impact on the magnetic properties. Naturally researching magnet compositions could be done via experimental work, however apart from it being rather time consuming, it would also be very expensive as you have to build new magnets from scratch and test them. Instead of experimental work one could opt to do it using micromagnetic simulations [FN00, SF99] which would get rid of material costs as well as the time used for making the magnets. However the simulations, depending on the system size, can be time expensive themselves. The idea is to supplement the micromagnetic simulation with predictions made through a neural network model. [EFK⁺18, KEK⁺24] By feeding the neural network with already made simulations we want to predict properties of magnets that were not calculated yet. Through this we can filter through different microstructures and do calculations only on promising microstructures. A relatively new idea in this regard is graph based machine learning also called graph neural networks, that seeks to transform the given data into a graph form to include structural information, such as grains and their neighbours, directly in a model and utilise its structure via message passing to predict properties of the system. [HCLB23]

The goal for this thesis is to verify the viability of graph neural networks in regard to predicting the properties of permanent magnets via classifying the coercivity through binary classification tasks and will be structured as following. The second chapter will lead us through physical properties of magnets and how magnets are simulated, with the functionality and theoretical aspects of neural networks and graphs right after in chapter three. To finish the theoretical aspect of this thesis a general overview of the data, where it comes from and how it is analysed, used and transformed will be given in chapter 4. Finally, in chapter 5 we will look at different datasets with varying structural properties and train a graph neural network on each of them. We will measure the neural networks performance according to common benchmarking tools such as confusion matrixes and accuracy, to find a baseline for which graph neural networks can be used and try to predict the coercivity of grain structures bigger than the grain structures fed to the neural network.

2 Magnets

In this chapter we will learn about magnets, how they are made, which properties they have and how they can be simulated.

2.1 Making magnets

While magnets can occur naturally they are not of interest for us as they are inferior to artificially made ones, which can be made through various construction methods. The different methods can be viewed in [COP⁺22a] and the follow-up article [COP⁺22b] which go into detail on how magnets are made. For our purposes we only need the first few steps which conveniently are the same across all manufacturing processes.

First we have to choose the materials the magnet will be made of. At the moment the strongest permanent magnets are made of a mixture of neodymium, iron and boron. As oxygen contamination (via oxidation) is highly detrimental regarding the magnetic properties of materials it seems logical to evacuate the production environment from all contamination sources. This has to be done by exchanging the air in the furnace with e.g., argon which eliminates oxidation and moisture at the same time. Alternatively the whole process could also be done in a vacuum, but in general it is easier to fill a chamber with argon than to hold a vacuum and making sure that the last few per mill of air left are so small so that no oxidation can occur. Next the cast metal gets crushed for example in a hammer mill, followed by ball milling or jet milling to achieve a small particle size. Then the powder is mixed with a neodymium rich dust significantly smaller than the particles to fill empty spaces. The empty spaces could occur in the next step of pressing the mixture in the presence of an external magnetic field. The external field has the purpose of aligning the particles, henceforth grains, according to their magnetic axis. Although there are additional processes after the pressing and sintering, such as grinding and plating, they would not give us additional information regarding the topics discussed in this thesis and will therefore be skipped, but if there is interest we refer to the two above mentioned articles. Additionally the first few steps can be viewed in Figure 2.1.

2.2 Properties of magnets

As we get data of magnets we have the need of understanding key parameters and their importance. For this we will primarily use [Coe10] to describe some of the core concepts of magnets and magnetism.

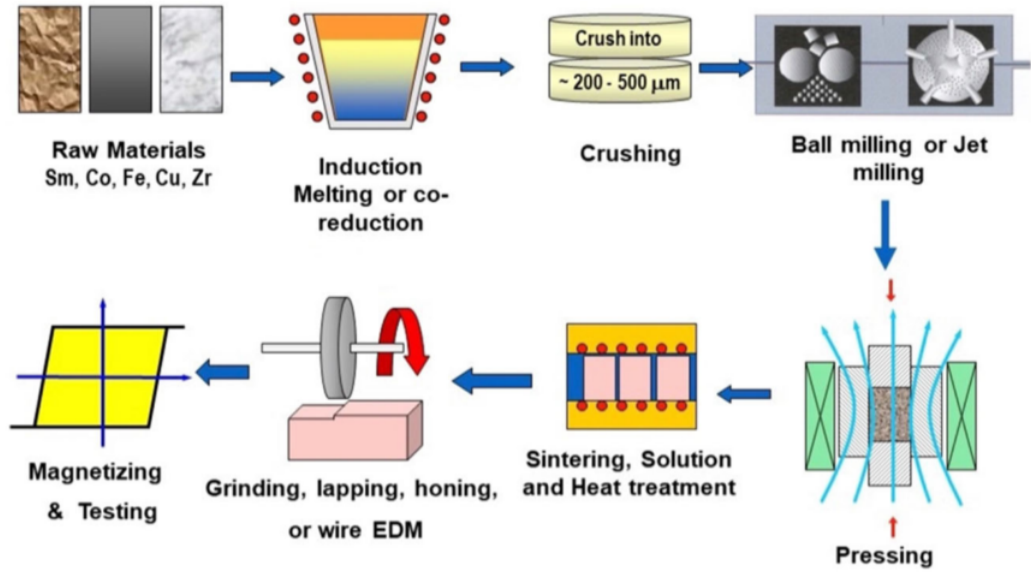


Figure 2.1: We can see the general production process as described in this chapter. Beginning from the raw materials to the finished magnetised magnet. [COP+22a]

2.2.1 Hysteresis and coercivity

Any ferromagnetic material has a particular response to an external magnetic field giving it a characteristic, which can be recorded as the hysteresis loop. The loop itself is done through a testing process where the material starts demagnetised and an external field is gradually increased. This magnetises the material in one direction until a saturation point is reached (M_{sat}), when the external field is then turned off we can see the remanence (M_r). When we then gradually increase the external magnetic field again but this time in the other direction we can find the coercivity (H_c). The coercivity is the field needed to demagnetise the magnet or also the point where the magnetisation switches signs. In Figure 2.2 we can see some of the described points and how a magnet might look like at these points.

2.2.2 Magnetocrystalline anisotropy and easy-axis

Most permanent magnets have a preferred magnetisation direction caused by their crystal structure. In this work magnets with a tetragonal crystal structure are investigated as they show a uniaxial anisotropy. The magnetisation lies along a preferred crystallographic direction. This direction is called easy axis. The work required to rotate the magnetisation out of the easy axis is the anisotropy energy. Since there is a certain distribution of easy-axes it is not possible to align an external magnetisation field perfectly to each and every easy-axis of the grains, therefore meaning that the resulting magnetisation may not

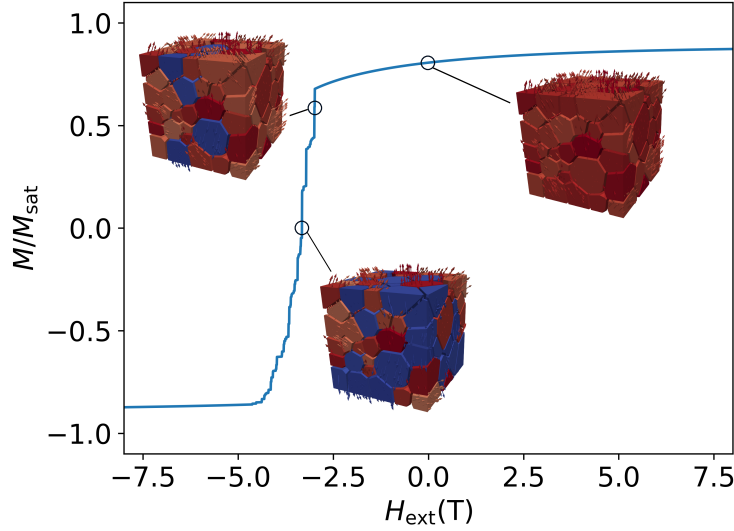


Figure 2.2: In this figure we can see the different stadia a magnet runs through during a demagnetisation process. In particular we can see how a magnet looks like at the remanent point without an external field, when the first grains switch and finally when the coercivity point is reached where the magnetisation is zero.

be in the preferred direction. This relation can be seen in following formula:

$$\epsilon_a = K_u \sin^2 \theta \quad (2.1)$$

$$E_a = \int K_u \sin^2 \theta d^3r \quad (2.2)$$

Here ϵ_a is the anisotropy energy density, E_a the anisotropy energy, θ the angle between the direction of magnetisation and the easy-axis and K_u the anisotropy constant. Another equally important aspect seen from this formula is that flipping the magnetisation by 180 degrees will lead to the same energy. Meaning that we have maxima at 90° and 270° as well as minima at 0° and 180° . Each grain is a single crystal and has its own easy axis. The easy axes of the grains are not perfectly aligned. To reverse the direction of the magnetization of a grain the magnetization requires work, as the magnetization has to rotate out of its preferred direction. In this thesis we only look at magnets with tetragonal crystal structure with positive K_u . Then we have the advantage of one easy axis, because depended on the sign of K_u and the crystal structure there can be more than one axis or even easy-planes. Anisotropy can also be seen as a limiter of coercivity in hard magnets which can be seen when we also take the magnetic constant $\mu_0 = 4\pi \cdot 10^{-7} \text{ (N/A}^2\text{)}$ as well as the spontaneous magnetisation $M_s \text{ (A/m)}$ into consideration:

$$H_c < 2K_u / \mu_0 M_s \quad (2.3)$$

In this formula the shape anisotropy is not considered and it is shown that the anisotropy is meant as the upper limit of the coercivity only lowered by the spontaneous magnetisation as well as the magnetic constant.

2.2.3 Demagnetising energy

Each magnetic dipole in a magnet produces a dipole field influencing each and every other dipole inside the magnet. Therefore the demagnetisation energy is also called dipolar interaction energy and as it influences each dipole it is considered to be a long range interaction. Inside the magnets it is also called the demagnetising field and outside the stray field. [ESS20] The formula for it is given by [Coe10] p.237]:

$$E_{\text{demag}} = -\frac{\mu_0}{2} \int \mathbf{H}(r)_{\text{demag}} \mathbf{M}(r) d^3r \quad (2.4)$$

Here $\mathbf{H}(r)_{\text{demag}}$ is the demagnetising field, $\mathbf{M}(r)$ is the magnetisation as the sum of atomistic magnetic moments per a small control volume and $|\mathbf{M}| = M_s$.

2.2.4 Exchange constant and magnetic polarisation

Describing the spontaneous magnetic polarisation is quite easy as it is simply just the spontaneous magnetisation M_s multiplied with μ_0

$$J_s = \mu_0 M_s \quad (2.5)$$

The exchange constant or also exchange energy in turn requires a little digression into the field of quantum mechanics. Inside of magnets there are also magnetic spins aligned, preferably, in the same direction. These spins are not independent from another but affect each other trying to align the direction of the other spins their own direction and vice versa. Within the framework of micromagnetism [BJ59] we describe magnetization processes with the continuous vector field $\mathbf{M}(x)$, whose direction is changing with position. Starting from the exchange interactions between the magnetic moments that are associated with the atoms in a ferromagnetic material, a continuous expression for the exchange energy can be derived [Aha00]:

$$E_{\text{exch}} = \int A((\nabla e_x)^2 + (\nabla e_y)^2 + (\nabla e_z)^2) d^3r \quad (2.6)$$

Where $\mathbf{e} = \mathbf{M}/M_s$, A is the exchange stiffness constant which is a function of the quantum mechanics exchange integrals and the lattice constants.

2.2.5 Zeeman energy (external field energy)

The Zeeman energy describes the effect an external magnetic field has on the magnet if it resides in it and therefore affecting its magnetisation. It is given by the following formula:

$$E_{\text{ext}} = - \int \mu_0 \mathbf{H}_{\text{ext}} \mathbf{M} d^3r \quad (2.7)$$

$\mathbf{H}_{\text{ext}}$ describes the external magnetic field. [ESS20]

2.3 Models and simulations

As so often is an analytical approach while simulating magnets time expensive or just not viable. Therefore with the development of computers soon a computational solution was proposed in the form of numerical treatments of micromagnetic equations. Starting with limited calculative resources the first approach was static i.e. time independent and done by minimising the total energy of the system with respect to the direction of the unit vector $\mathbf{e}$ of the magnetization [Coe10, p.234]:

$$E_{\text{tot}} = E_{\text{exch}} + E_{\text{A}} + E_{\text{demag}} + E_{\text{ext}} \quad (2.8)$$

The respective formulae for the various energy terms can be found in the previous section with the integral versions as the continuum model. We then have to further differentiate the two approaches used in numerical solutions, namely finite difference and finite element methods. Whereas the finite difference method uses an regular grid to approximate differential operators with finite differences, the finite element method expands with linear basis functions on a tetrahedral finite element mesh. For computing the demagnetizing field the magnetostatic boundary value problem needs to be solved. within the framework of the finite difference method [MD⁺07] the field is computed by summing up the magnetic field sources, accelerated by the use of a fast Fourier transformation. Within the framework of the finite element methods, the partial differential equation for the magnetic scalar potential is solved and the field is derived from its negative gradient. Each of these methods have their respective advantage. Given a simple magnet the finite difference method will need less computational resources, but in complex magnets the finite element method has the advantage of capturing the shape of the granular structure more precisely. [Öze18, p.20]

2.3.1 Stoner-Wohlfarth model

One of the earliest model of magnetization reversal of permanent magnet is the Stoner-Wohlfarth model. It assumes that the magnetization of the particles of the magnetic materials remain uniformly magnetised during magnetisation reversal. This assumption is fulfilled for small particles of ellipsoidal shape which are often called Stoner-Wohlfarth particles. [Coe10, p.247] The energy density can then be written as

$$E_{\text{tot}} = K_{\text{u}} \sin^2 \theta - \mu_0 M_{\text{s}} H \cos(\alpha - \theta), \quad (2.9)$$

where we can obtain one or two energy minima by minimising with respect to θ . The switching field of a Stoner-Wohlfarth particle is given by [GOF⁺20]:

$$H_{\text{SW}} = 2K_{\text{u}}/J_{\text{s}} \left(\cos(\alpha)^{2/3} + \sin(\alpha)^{2/3} \right)^{-3/2} \quad (2.10)$$

2.3.2 Reduced order model for permanent magnets

The reduced order model described in [MKF⁺24] and developed in [EFK⁺18] uses the SW model in micromagnetic simulations by placing many SW particles slightly under the surface of the grains, not their inner parts. So when an external field is applied it is checked whether this external field is higher than the necessary switching field described in eq. 2.10. If that is the case for one of the SW particles in a grain, the magnetisation for the whole grain switches signs. If the magnetisation of a grain is reversed, the demagnetisation field $\mathbf{H}_{\text{demag}}$ has to be calculated again. This process is repeated until all the grains in the magnet switched signs, so that one side of a hysteresis loop is recorded. A depiction of where the SW particles are and how a switch may occur can be seen in Figure 2.3

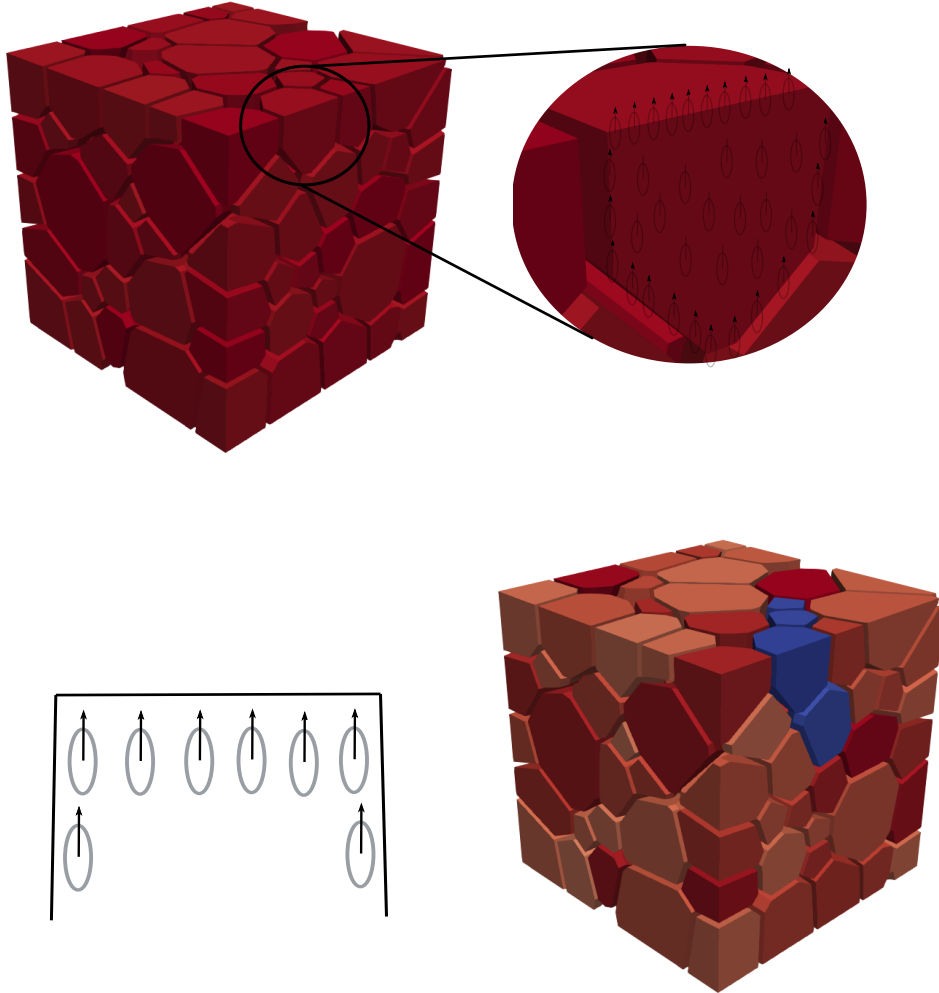


Figure 2.3: A visual description for the usage of the Stoner-Wohlfarth model is used in micromagnetic simulations. As can be seen from the upper images, the SW particles are in each grain beneath the surface, oriented along the easy-axis. In the lower pictures we can see again that the particles are only on the surface not in the inner parts of the grains. If the magnetization in one of the embedded Stoner-Wohlfarth particle switches, the magnetization of the corresponding grain will be reversed. A reversed grain will increase the magnetostatic field in its neighbouring grain below. Subsequently also this grain will switch. Such a switching cascade is shown in the bottom right figure.

3 Graphs and neural networks

Now we will look at graphs and neural networks i.e. what are graphs and how they are used in the context of machine learning and neural networks. We will follow [BHB⁺18] as well as [SLRPW21] first in order for us to understand what graphs are, switch to [Gér22] in between to learn about neural networks, and go back to [BHB⁺18] to finish with how graphs utilise neural networks.

3.1 Graphs

Graphs are relational structures built by two different parts, namely nodes and edges. Edges in particular can hold additional information regarding the direction of information differentiating between directed and undirected connections. In molecules for example we can take the atoms and model them as nodes and the bonds between them as our undirected edges of a graph. This alone already counts as a graph, but we would lose important information regarding the molecule. Therefore, every node and every edge get assigned an arbitrary number of respective properties. In machine learning, these properties are called node and edge features. Additionally, the whole graph can hold global features, which represent properties at the molecule level. In our molecule example we can give different atoms numbers (H=1, C=2, ...) and deposit this information in the specific nodes in the graph. For the edges the same thing can be done in regard of which type of bond connects the atoms. Finally for the global feature we may specify the acidity or other categories the molecule as a whole may have. We can see this done in Figure. 3.1 where a Caffeine molecule is transformed into its graph equivalent form. Molecules already are some kind of graph as the difference between the two pictures lies mainly in visualisation. Additionally we can also see that graphs do not have to adhere to structural constraints as the benzene rings do not hold their form and also the angles the bonds can have are not of importance any more in graph form. However, the bond angle could be added as node or graph features.

3.1.1 Usage of graphs

Now that we know what graphs are we can take a look at for what they are used for. They are used for three different types of problems namely node, edge and graph classification or regression tasks.

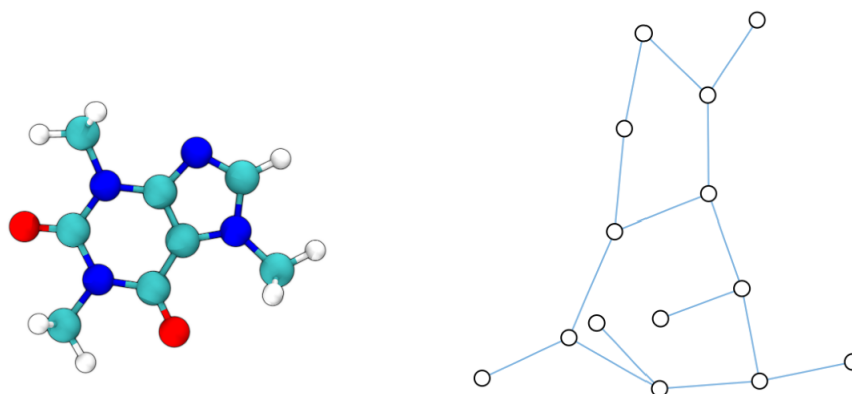


Figure 3.1: Here we can see what the molecule on the left (Caffeine) would look like in its graph equivalent form. Whereas the molecule view on the left shows accurately which type of element is at which place in the molecule the right side hides this in the node features. This can also be seen with the hydrogen bonds (the white spheres) and in general the bond types, on the right side all these visual features are put into either the node features (number of hydrogen atoms bonded) or the edge features (type of bond). [\[SLRPW21\]](#)

Node classification

Lets begin by taking a look at node classification. As the name suggests it predicts the state of individual nodes contained in a graph. The most common used example for this task is Zach's Karate Club [\[GN02\]](#). The example consists of a graph depicting a social network of Karate students and teachers (nodes). The lines connecting the nodes mirror their interaction outside of the club. When the club splits into two we have a task of classifying the students according to their connections and labelling them to their new teacher. We can see this task done in fig. [3.2](#)

Edge classification

The next classification tasks are edge classifications. This means the task is to predict the connection/strength of the connection or the features of edges between nodes. Lets say we have a number of publications given with different categories the task would be to predict if these different publications are connected via citations (edges). Another example would be movie and series recommendation with respect to the genres you watched till now or also friend recommendations in social networks as you may be a friend of a friends friend. In the movie and series example the movies are the nodes with their genres as node features giving the edge prediction similar movies to suggest you. In the friend recommendation example each friend is a node with each node connected to friends they have added in their social network. The edge prediction suggests you friends of friends as

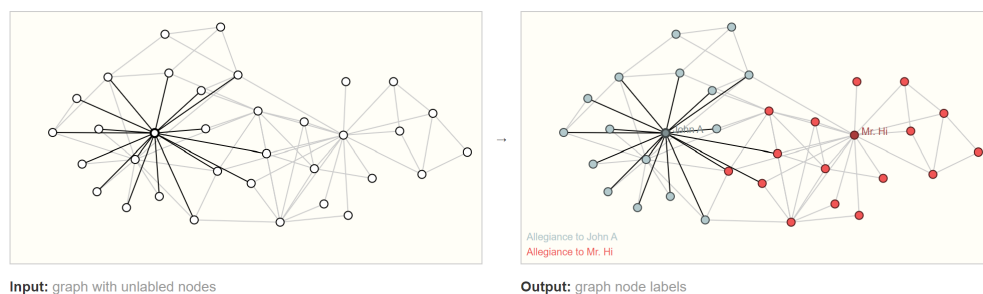


Figure 3.2: An overview on how the task of node classification is handled in the karate club example. Given an unlabelled graph on the left and a labelled one on the right. [SLRPW21]

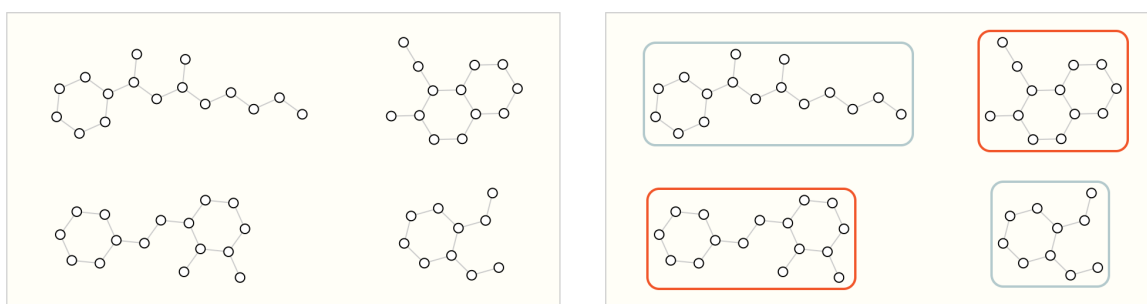


Figure 3.3: A graph classification task by counting the number of benzene rings. [SLRPW21]

the probability to know a friends friend is rather high, increasing even when more than one of your friends added the same person to their network. [WJ22]

Graph classification

Lastly we have graph classification tasks, where we predict global features of graphs by looking at its structure and components. We could for example classify molecules by their structure and predict if they are pungent or not. [SLWL⁺19] Other use cases include whether a molecule will bind to receptors for diseases or if it has benzene rings which can be seen in Figure. 3.3. This classification task is also used in this thesis as we want to predict the properties or rather strength of magnets by looking at their microstructure. Naturally this includes transcribing the microstructure into graph form and depositing the properties in the appropriate features e.g node, edge and global features.

3.2 Neural networks

After we defined what a graph is and which different tasks exist for it we will now look at neural networks before learning how it is used in generating results in a graph neural network. We will follow [LZ20, p.13] in their approach.

3.2.1 Architecture

A neural network, given its name, is modelled after the network of neurons in the brains of animals including our own species. Similarly the smallest parts of a neural network are also called neurons, computational units which are connected to another. It receives n inputs $\{x_1, x_2, \dots, x_n\}$, including their respective weights $\{w_1, w_2, \dots, w_n\}$, and a bias term b to finally run the weighted sum of these terms through an activation function resulting in an output z seen in eq. 3.1. This output, typically, ranges between 0 (fully deactivated) and 1 (fully activated) and is used as input for upcoming neurons.

$$z = f\left(\sum_{i=0}^n w_i x_i + b\right) \quad (3.1)$$

Commonly used activation functions include:

- Sigmoid Function

$$\sigma(x) = \frac{1}{1 + e^x} \quad (3.2)$$

- ReLU

$$\text{ReLU}(x) = \begin{cases} 0 & x \leq 0 \\ x & x > 0 \end{cases} \quad (3.3)$$

- Tanh Function

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \quad (3.4)$$

The tanh function sets itself apart from the other as its results can range between -1 and 1. The functions and their course can be seen in Figure 3.4.

Now in contrast to the biological counterpart the brain, are the neurons in neural networks not disorganised, but ordered in layers. The first layer is the input layer as it receives the initial data from a dataset and the last layer is the output layer giving a result. The input layer for the neurons is done via embedding which is a linear combination of the input data. Through this linear combination can the original input be transformed to any size through the weight matrix. Every layer in between is called a hidden layer and there is no restriction on how many there can be. However their quantity defines the

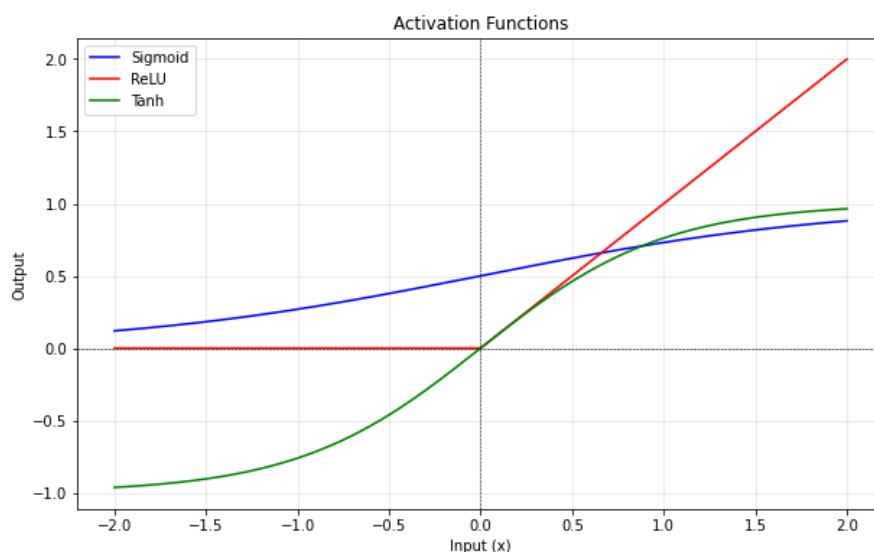


Figure 3.4: A plot of the different activation functions showing their courses.

deepness of a neural network as a network with only one hidden layer is shallower than one with three.

Choosing the right amount of hidden layers and neurons per layer is a task in itself. Following [Gér22, pp.324] we know that one hidden layer may suffice, as a neural network with one hidden layer can accurately model complex functions. This beckons the questions why we even would want to go deeper. The answer to that is efficiency, a deep neural network has a higher parameter efficiency meaning that they can reproduce the same results with fewer neurons as a neural network with one hidden layer. Each added hidden layer learns more complex structure from the data. So, it is not only more efficient, but also shown to converge faster and generalise better. One should still begin with one or two hidden layers as most functions can be modelled by them and gradually increase their number if necessary. The network should also not be too deep as one will run into overfitting which is looked at in more detail in Section 3.2.3. Given the number of layers we also have to address the number of neurons per layer. This is rather easy for the input and output layer as the number of neurons in the output layer is governed by the task on hand. If it is a regression task there will be only one neuron and if it is a classification task the number of neurons will mirror the number of different classes. Similarly it is done in the input layer, as here the number of neurons can be equal to the number of input features. Lets say we have a image dataset that we want to train on, then the number of neurons is equal to the number of pixels in the images e.g., 50x50 pixels means 2500 neurons. Lastly we have to address how many neurons should be in the hidden layers. It was standard practice to form a pyramid, meaning that the hidden layers subsequently have fewer neurons than the ones before them. For example the input layer of our image example has 2500 neurons, then the next layer has 2000 neurons and the one after that

1000 neurons, forming a sideways pyramid. The reasoning behind forming this pyramid is that the layers learn different types of complexity from the dataset. The first layer learns low level complex structures while the later layers learn high level complex structures and therefore need lesser neurons. However this was tested and shown that an equal amount of neurons per hidden layer performs as well or even better than a pyramid structured neural network with the added benefit that there is only one hyperparameter to tune instead of many hyperparameters depending on the number of layers. One should add however that not reducing the number of neurons per layer leads to more training effort as more parameters are introduced to the neural network.

3.2.2 Classification or regression task

Apart from firstly thinking if the dataset we want to research is viable to be put into graph form, we also have to think about which results we want to get. Here we can differentiate between either a regression or a classification task. For a classification task the dataset is subdivided into n different classes. This is for example the case with a dataset of images where we want to differentiate galaxy types by their phenomenology i.e number of spiral arms, curvature or central mass. This would lead to a model that classifies an image into multiple different classes [GD22]. On the other hand when we look at our graph classification example from section 3.1.1, where molecules were classified according to their number of benzene rings, we can see a binary classification task. The regression task however tries to find a function that mirrors the dataset so that we can predict results for previously untested points according to this function. For example if we have a dataset of magnets and their coercivity we want to predict the coercivity of new magnets according to similarities between the magnets. Graph neural networks in particular were shown to generalise well not only in systems of the same size but to bigger size systems as well. In [SGGP⁺20] for example it was shown that a graph neural network trained on fluid dynamics could predict the dynamics of a system that is one magnitude bigger than the trained system sizes.

3.2.3 Training

Training describes the learning process a neural network goes through in which it adjusts the weights and biases connecting the different neurons to another. Learning is mainly done via two different methods called supervised and unsupervised learning. Supervised learning describes the process where the dataset on which the neural network is trained on is labelled. This means that every object in the dataset has a clear solution given to the network. On the other hand unsupervised learning does not provide a label and the network is expected to correctly classify similar objects through pattern recognition and grouping them together. Other training methods include reinforcement learning as well as semi-supervised learning. [RN10, p.671]

We have mentioned the training of the neural network on a dataset. In the following, we briefly describe this training process. During the training process the dataset we want to research is split into three sets. These sets are called training set, validation set

and test or generalisation set. The training set is the bulk of the available data with around 70% of the original dataset content and intended for the neural network in the training step to learn from it. The set size is therefore chosen as high as possible so that the neural network can learn diverse data. Next the validation set is around 15% of the original dataset size and is used in the training step to alter the hyperparameters of a neural network so that it can accurately guess until now unseen data. This means that this set is not used for training but fine-tuning in the training step. The test set is again around 15% of the original dataset size and used after the fine-tuning is finished to check the finished model on till now unseen data and benchmark it. [RN10, p.684] Finally, as hyperparameters were mentioned, they are different settings one can adjust before the training starts. Hyperparameters include the depth of the neural network, the number of neurons, the learning rate and much more depending on which neural network is used. So how does a training loop actually look like after setting all these parameters? A training loop commonly consists of 4 steps:

1. From the training set a number of samples is chosen, including their true values y_{true} , depended on the learning type.
2. Run the model on the chosen samples to get the predicted results y_{pred} .
3. Evaluate the loss of the model through the chosen loss function.
4. Update all the weights and biases in the neural network so that the loss function gets minimized.

The loop is repeated several times. A full path through all samples of the training set is called epoch. Epochs are repeated until a loss criterium is satisfied or the loss in the validation set shows no more improvement.

Loss function

The loss function is used in the training process as an error of the model and training a model means minimizing this loss function. Alternatively it can be called a function that defines "the amount of utility lost by predicting $h(x) = y_{\text{pred}}$ when the correct answer is $f(x) = y_{\text{true}}$." [RN10, p.669]. The loss function is defined by $L(y_{\text{true}}, y_{\text{pred}})$ and $L(y_{\text{true}}, y_{\text{true}})$ is always zero, since the correct value was predicted. Then we can further differentiate between use case, as regression models need different loss functions compared to classification models, and how we rate the errors i.e. how harshly we want to learn from errors. In the regression case this can be done via the absolute value loss function

3 Graphs and neural networks

(L_1) or alternatively the squared error loss function (L_2):

$$L_1(y_{\text{true}}, y_{\text{pred}}) = |y_{\text{true}} - y_{\text{pred}}| \quad (3.5)$$

$$L_2(y_{\text{true}}, y_{\text{pred}}) = (y_{\text{true}} - y_{\text{pred}})^2 \quad (3.6)$$

$$J(\theta)_{\text{MAE}} = \frac{1}{m} \sum_{i=1}^m |y_{\text{true},i} - y_{\text{pred},i}| \quad (3.7)$$

$$J(\theta)_{\text{MSE}} = \frac{1}{m} \sum_{i=1}^m (y_{\text{true},i} - y_{\text{pred},i})^2 \quad (3.8)$$

From the equations we can also see that the L_2 loss function places a greater importance on the accuracy of the prediction. This can be inferred if we think about the error sizes. In the case of a big error the L_2 function makes the error even bigger punishing the neural network harder than the L_1 function. In the case of a small error the reverse is the case as the L_2 function makes the error even smaller. The loss of the individual samples can be summed up as seen in the formulae for J_{MAE} and J_{MSE} . If we want to minimize the error rate of classification tasks we can use the cross entropy loss function:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y_i \log(p_i) + (1 - y_i) \log(1 - p_i)] \quad (3.9)$$

With p being the probability of the respective values, y_i the label of the sample and θ being the parameter vector containing the feature weights previously mentioned as w . For classification tasks, the neurons of the output layer give the probability to be in a certain class. [Gér22, p.144]

Gradient descent and variations

We mentioned that during the training process we want to minimize the loss function to then update the weights associated with each neuron. This can be done through a method called Gradient Descent which utilizes derivatives to find global minima. The idea is to look at where you are standing, as in if you stand on a mountain you can feel the slope and its direction. Through this feeling you can find the way down into the valley and deepest point. Gradient Descent does exactly that as "it measures the local gradient of the error function with regard to parameter vector θ , and it goes in the direction of descending gradient." [Gér22, p.118]

$$\nabla_{\theta} J(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} J(\theta) \\ \frac{\partial}{\partial \theta_1} J(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} J(\theta) \end{pmatrix} \quad (3.10)$$

Here we can see that we build the gradient of a cost function via partial derivatives of the model parameters i.e., weights and biases. This vector points uphill. To find the

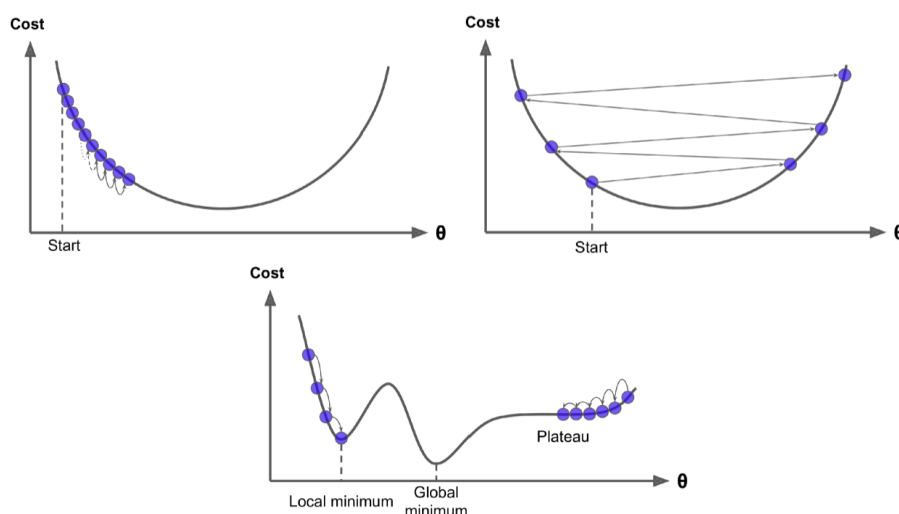


Figure 3.5: A visualisation of the issues gradient descent can run into. The top left shows a too small step size, top right a too large and the bottom pictures shows the issue of local minima. [Gér22, p.119-120]

minimum we therefore have to subtract it from the weights vector after multiplying it with its step size. This optimization procedure is highly susceptible to its step size, which is controlled by the hyperparameter learning rate. Suppose the learning rate is too small then the convergence to a solution will also take more time, and if the step size is too big you might skip over the metaphorical valley and stand on another mountain, missing the solution completely. Another mistake, that can happen during the optimization process, is descending upon a plateau or a local minimum instead of the global minimum. The effect of step size as well as the local minima and plateau can be seen in Figure. 3.5

As each input parameter is an additional dimension that has to be considered when finding the global minimum, it is necessary to scale them appropriately. This can help with the convergence rate. **Batch Gradient Descent** is a variation of Gradient Descent, which considers the whole training set when taking the gradient of the current model and therefore highly depended on the training set size. If the training set is too large the convergence rate in turn will be slower as well. For finding a local minimum, it would be possible to use the entire training set for building the gradient. However, here we want to find a global minimum. Therefore variants of the method, that may escape local minima have been introduced under the name of **Stochastic Gradient Descent** and **Mini-batch Gradient Descent**. The main difference in their approach is the set size that is considered in each step. When Batch Gradient Descent considers the whole training set during one step, Stochastic Gradient Descent is doing the opposite and only considers one element of the training set when calculating the partial derivative of the cost function. Mini-batch Gradient descent uses a small set of random elements from the training set for the gradient. Both of these algorithms converge faster but move around

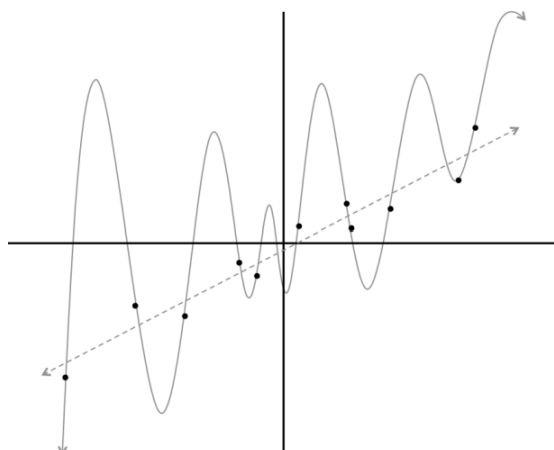


Figure 3.6: A visualisation on how a model might approximate the given data with the linear function as solution and the polynomial function the overfitted model. [BBP22]

more erratically meaning that their final result is good but not optimal. This can be helpful when finding the global minimum as they will jump out of local minima, but they will never settle in the global minimum. To circumvent this we can implement a learning schedule, that changes the step size after a set amount of iterations. [Gér22, pp.121]

Over and underfitting

During the training step we can run into several issues, some of which are known as over and underfitting. Overfitting can be described as a generalization issue as in the model predicts the training set and validation set perfectly but has issues with the test set. This happens because the neural networks learns noise or unnecessary patterns in the training set which may not be mirrored in the test set. We can also see this in Figure 3.6 where two models describe the given data. The counterpart to overfitting is underfitting where the model cannot learn the nuances of the provided data. We can see this if the validation set as well as the test set are not predicted well. To counter overfitting we can fall back on regularization techniques. For underfitting in turn we can complicate our model as in increasing the number of layers and neurons, train for more epochs, give better features to the model or increase the training set size. [Gér22, p.27]

Regularization

In the last section we briefly talked about ways to circumvent over and underfitting. We will now look deeper into techniques to counter overfitting. Beginning with **early stopping** we have a technique that limits the amount of epochs through different measures but mostly according to loss. The hyperparameter introduced through early stopping is patience which means essentially if the model does not have a smaller loss after a certain

amount of epochs the training will be stopped.

Next we have **weight regularization** specifically know as L1 lasso and L2 regularization. These two are ways to constrain the weights of the neural network to specific values. The formulae of the two are given by:

$$J(\theta) = COST(\theta) + \alpha \sum_{i=1}^n |\theta_i| \quad (3.11)$$

$$J(\theta) = COST(\theta) + \alpha \frac{1}{2} \sum_{i=1}^n \theta_i^2 \quad (3.12)$$

With the first equation being L1 regularization and the second L2. The L1 regularization is used in models where its suspected that there are features that are less relevant than others, because it tends to set the weights of the least important features to zero. The L2 regularization function is used so that the weights are kept as small as possible, another name for it is weight decay. There is a third weight regularization technique called elastic net regularization which combines the other two into one. The hyperparameter introduce in this technique is α also know as regularization strength. [Gér22, pp.135]

Another technique is **dropout** which introduces a mechanism during training that temporarily deactivates neurons during one training step. This can include input neurons as well. The hyperparameter introduced with this technique is called dropout rate p with a typical value between 10% and 50%, which tells the neuron the probability with which it will be deactivated for this step. This technique is meant to make neurons independent from another and improves generalisation. [Gér22, p.365]

3.2.4 Benchmarking

After the training is completed we have to check how good a model is. Again we have to differentiate between regression and classification tasks as their benchmarking is different. For **regression tasks** we can choose from a variation of mean squared error functions. We will use a slightly changed notation from [PSBBS22] for which N is the size of the dataset, y_{pred} is a $N \times 1$ vector with the values of the predicted outcome for each entry in the dataset and y_{true} is the same shape as y_{pred} but with the true values corresponding to the samples in the dataset.

The **Mean Absolute (Gross) Error** is measuring the average error over all predictions and their real values given by:

$$\text{MAE} = \frac{1}{N} \sum_{i=1}^N |y_{\text{pred},i} - y_{\text{true},i}| \quad (3.13)$$

Another variation is given by the **Mean Squared Error** which replaces the absolute value by squaring the sum:

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^N (y_{\text{pred},i} - y_{\text{true},i})^2 \quad (3.14)$$

3 Graphs and neural networks

This however makes it prone to outliers meaning that a result that significantly deviates from the true value gets even bigger and therefore skews the whole average. A simple solution for this is given by the **Root Mean Squared Error**:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{N} \sum_{i=1}^N (y_{\text{pred},i} - y_{\text{true},i})^2} \quad (3.15)$$

which additionally has the benefit that the errors have the same units as the predicted and true values. It should be mentioned that the coefficient of determination R^2 is deliberately not explained as it is only suitable for linear regression tasks and should not be used for non-linear regression tasks according to Spiess and Neumeyer. [SN10]

For **classification tasks** we use a variety of accuracy measures to benchmark the model and follow [Gér22, pp.90] in their approach. In the simplest case we measure the accuracy i.e. how many percent of the different sets were correctly classified. This can be extended to the next benchmark, the confusion matrix. The confusion matrix has on the main diagonal the true predicted values as in true negative (TN) and true positive (TP) and on the off diagonal the false positive (FP) and false negative (FN) values. This is the case for binary classification if there are more cases we generally expect the same structure with true predicted values on the main diagonal and on the off diagonal the falsely classified values depended on case. From the (binary) matrix we can derive two separate measures of quality the precision and recall given by the following formulae:

$$\text{precision} = \frac{TP}{TP + FP} \quad (3.16)$$

$$\text{recall} = \frac{TP}{TP + FN} \quad (3.17)$$

The precision defines the accuracy of the positive predictions and recall is the true positive rate or alternatively the ratio of correctly detected positive values. We can see how a confusion matrix might look like in Figure 3.7

It is convenient to combine these two measure via the F_1 score:

$$F_1 = \frac{2}{\frac{1}{\text{precision}} + \frac{1}{\text{recall}}} \quad (3.18)$$

This measure is higher if precision and recall are both high. In some classifiers this is not wanted as a higher precision is needed e.g. in child filters where we want to correctly classify every page as child safe and do not want any false positives (bad pages deemed child safe) which in turn also means we may classify child safe pages as bad (false negatives) giving a bad recall. In reverse for spam filters we want a high recall as we do not want any spam to get through (false negatives) and do not care if some mails for example get classified as spam even though they are not (false positives). This is known as the precision and recall trade-off.

Lastly we have a variation of precision and recall with the **ROC curve**. This metric plots the recall, we remember it is also known as the true positive rate, against the false

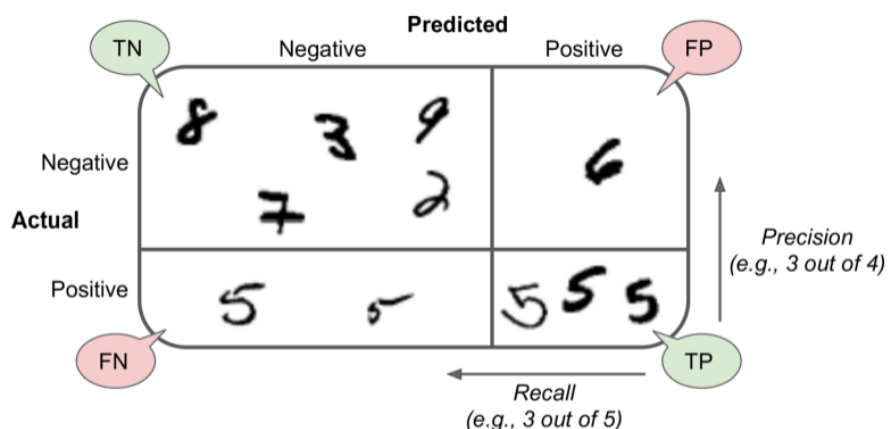


Figure 3.7: Confusion matrix showing the quarters and what they mean using the MNIST dataset where written numbers are classified. [Gér22, p.92]

positive rate. We want the curve to stay in the upper left corner and it is commonly used to compare different classifiers via the **area under the curve (AUC)**. If the ROC AUC metric score is equal to one a perfect classifier was trained, is it however 0.5 we know that the classifier is simply guessing. It should be mentioned that the precision / recall curve is a better option if there is importance in false positives or if the positive values are rare. In Figure 3.8 we can see how the two curves might look like. Where we can also see that there is a trade off between precision and recall as increasing precision reduces recall and vice versa.

3.2.5 Calculations on graphs

Now that we have introduced what graphs are and how they are used as well as key aspects of neural networks we will look at how calculations on graphs are done so that a result is generated. For this we will use [BS24] as well as [LZ20].

Message passing

If we think about graph neural networks and what makes them special, message passing is on top of the list. It is a unique method which, in short, enables information from nodes and edges to be passed to their neighbours. Intuitively we may say that this is comparable to a convolutional neural network. Lets say we have a common neighbourhood represented as graph. The number of people per house are node features. Friendship between neighbours is encoded with edge features. In the first step a house receives a letter from every other house in its vicinity containing the node and edge features of them. The receiving house then sums up all the information, shortens it so that the amount of information stays constant and integrates it into its own information. This means that in the next iteration information from the surrounding houses is included in the letters as well. This corresponds to a message passing step in a graph neural network, with the

3 Graphs and neural networks

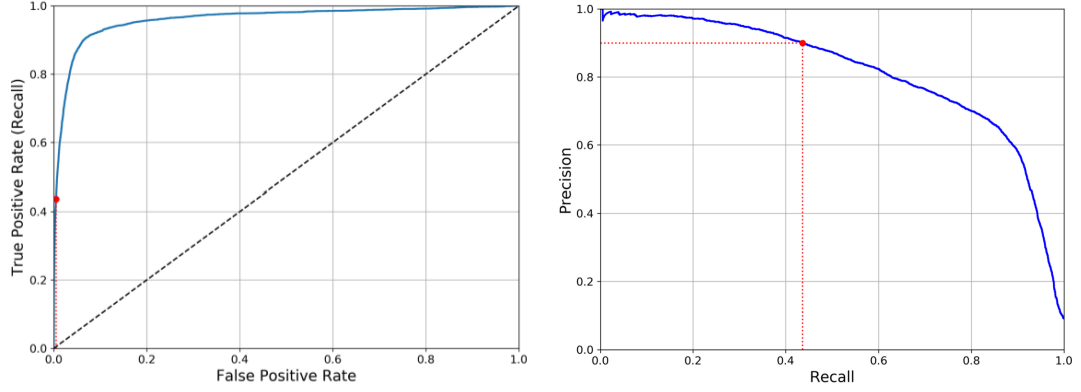


Figure 3.8: On the left we can see how a ROC curve might look like with the curve being in the upper left corner, suggesting a good model. On the right we have the precision over recall curve which helps understanding the model better.

[Gér22]



Figure 3.9: The message passing steps in order. First we have the input graph. Then the green node receives all of the neighbouring features, aggregates it through some invariant function. Then the message is transformed through the use of a neural network. Finally all the features in the graph get updated with their updates values. [BS24, p.23]

steps collection, aggregation, transformation and updating. The collection step, collects all features from neighbouring nodes, the aggregation step uses invariant functions like max or mean to sum up the content of the features, the transformation step uses the neural network to create new features (embeddings) for the nodes and finally the update step overwrites all initial node features to the newly generated ones. A graphical view of message passing can be seen in Figure 3.9.

Graph networks

As mentioned at the beginning of this chapter we will follow [BHB⁺18] to describe their implementation of a graph neural networks kit called graph networks (GN) and will supplement it with [LZ20, p.62] as they describe Battaglia et. al's graph network including its pro and cons. We adopt the notation from Battaglia et al..

The **graph architecture** of the GN is given by a 3-tuple $G = (\mathbf{u}, V, E)$ where $\mathbf{u}$ denotes the vector containing the global features, V the matrix $V = \{\mathbf{v}_i\}_{i=1:N^v}$ containing all nodes

feature vectors $\mathbf{v}$ with N^v being the number of nodes, E the matrix $E = (\mathbf{e}_k, r_k, s_k)_{k=1:N^e}$ containing all edge feature vectors $\mathbf{e}$ including the index of the receiver and sender node r and s with N^e being the number of edges.

Next we have the **neural network layer algorithm** as in what happens during each layer in the graph neural network, which is called **GN block** in both above mentioned sources. Such a block is built by six functions, three "aggregation" functions ρ and three "update" functions ϕ .

$$\mathbf{e}'_k = \phi^e(\mathbf{e}_k, \mathbf{v}_{r_k}, \mathbf{v}_{s_k}, \mathbf{u}) \quad \bar{\mathbf{e}}'_i = \rho^{e \rightarrow v}(E'_i) \quad (3.19)$$

$$\mathbf{v}'_i = \phi^v(\bar{\mathbf{e}}'_i, \mathbf{v}_i, \mathbf{u}) \quad \bar{\mathbf{e}}' = \rho^{e \rightarrow u}(E') \quad (3.20)$$

$$\mathbf{u}' = \phi^u(\bar{\mathbf{e}}', \bar{\mathbf{v}}', \mathbf{u}) \quad \bar{\mathbf{v}}' = \rho^{v \rightarrow u}(V') \quad (3.21)$$

In these formulae $E'_i = \{(\mathbf{e}'_k, r_k, s_k)\}_{r_k=i, k=1:N^e}$ depicts updated edges, $V' = \{\mathbf{v}'_i\}_{i=1:N^v}$ depicts the set of all updated nodes and $E' = \bigcup_i E'_i$ depicts the set of all updates edges. Every prime (') denotes a updated term and every overline denotes a aggregated term, which will be described in more detail or rather formula by formula later on. The $\phi^{e|v}$ update functions are calculating updates on each of their respective terms as in every edge and every node gets updates and the ϕ^u update function only gets triggered once to update the global features. The ρ aggregation functions take a input set seen in the brackets and transform them to one singular aggregated value containing all of the information given from the set. It must be mentioned that the ρ functions should not be swayed by reordering of their input values or changes in their sizes e.g., mean and maximum.

Computational steps done during a GN block flow commonly from edge updates to node updates to finally global updates as can also be seen in the order of the formula listed above. First we have the update function ϕ^e updating each edge according to its given arguments: the initial edge values, the two nodes that get connected via the edge and the global features. Then every updated edge gets an aggregated value through the aggregation function $\rho^{e \rightarrow v}$. Next every node is updated through the ϕ^v function taking all the updated neighbouring edge values, the nodes initial values as well as the global features into consideration. After this the node aggregation function $\rho^{v \rightarrow u}$ and the edge aggregation function $\rho^{e \rightarrow u}$ get triggered providing the aggregated value of the nodes and edges to the update function ϕ^u . The order of the updates does not have to be this way it can also be reversed and if only structural information shall be passed we can omit edge updates completely.

3 Graphs and neural networks

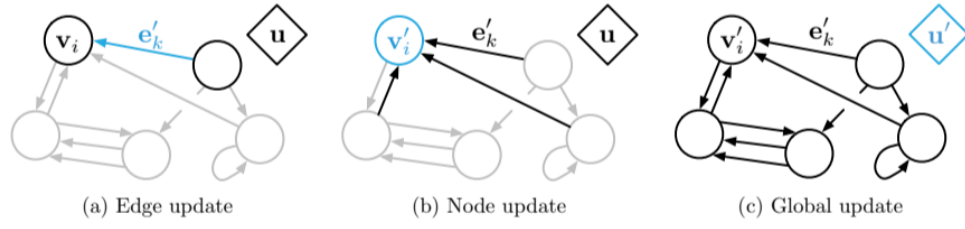


Figure 3.10: In this figure we can see the different update processes described in the computational steps paragraph, with (a) being the ϕ^e function (b) being the ϕ^v function and (c) being the ϕ^u function. The blue colour shows us which element is subject to change and the black colour shows us which other elements contribute in this update process. Gray means that it is not of relevance during this step. [\[BHB⁺18\]](#)

4 Data and graph representation

After covering the theoretical aspects we now look at the data which will serve as input for our graph neural network, how it will be transformed to a graph form and which preprocessing steps were done before training a model.

4.1 Data Sources and analysis

We mentioned that the simulations of magnets we do is done through an in-house developed reduced order model. This model has to be provided with the magnetic properties and structure before it can calculate the hysteresis loop. The structure of the magnet is generated through a program called Neper [\[nep\]](#), which fulfils the tasks of generating a magnet and its grain structure. To construct the graphs we are in need of the .tess tessellation file given from Neper which holds all the structural information regarding the magnet. We use Neper's analysis feature to extract geometrical information regarding grains as well as the graph structure from said file using `neper -T -loadtess tessellation file -statpoly keys`. The statpoly expression is for some keys exchanged with statess but for those specific keys it will be mentioned.

For statpoly we have the following keys:

- id - a identifier to differentiate between grains
- coo - the x, y and z coordinates
- diameq (unit of lengths) - the equal diameter a grain would have if it were a sphere
- sphericity - how similar a grain is to a sphere
- dianglemin (°) - the minimal dihedral angle of the grain
- dianglemax (°) - the maximum dihedral angle of the grain
- npolynb - number of neighbouring grains
- npolys - a list of neighbouring grain ids

And for statess we have:

- xmax - maximum system size in x - direction
- ymax - maximum system size in y - direction
- zmax - maximum system size in z - direction

4 Data and graph representation

The statts will be necessary if we want to transform the coordinates of the grains from cartesian to spherical coordinates. This is because the system is not centred around the origin, but is located in the positive x, y and z domain of the coordinate system. By subtracting half the maximum system size of each direction we achieve centring around the origin, which when converted to spherical coordinates, gives us an metric of how far the grain is away from the centre of the system. Apart from geometrical properties we are also interested in the magnetic properties of the system in question. There is a specific file which is prepared for the magnetic simulation so that the simulations has knowledge of the magnetic properties of the grains. This file is known as .krn file and holds the following information for grains that have magnetic properties:

- θ (rad) - polar angle of the easy axis in relation to the positive z-axis
- ϕ (rad) - azimuthal angle of the easy axis in relation to the x-axis
- K ($\frac{\text{J}}{\text{m}^3}$)- anisotropy constant of the grain
- J_S (T) - magnetic polarisation of the grain
- A ($\frac{\text{J}}{\text{m}}$) - exchange constant of the grain

And finally from a file called hc.dat we get the coercivity of the simulated magnets. These parameters are initially used and their importance for the graph neural network will be analysed. If they are not useful for training the network, because they have no effect on the result of the network or are actually detrimental on achieving a good model they will be removed.

4.1.1 Analysing of features

We look at variation and correlation of features through the so called correlation matrix. This matrix, built using the seaborn package [pyd], has on the main diagonal the features variation in form of a histogram. On its off diagonals we can see the correlation of features with other features. Naturally what we want to see is more than one bar in the histogram, as well as no complete correlation between the features as we could use one feature then to describe the other. Additionally we will look at the corresponding correlation matrix which measures the correlation between values by assigning a value between -1 and 1. If the correlation value is 1 we know that the values have a positive correlation and with -1 a negative correlation. The values are completely uncorrelated if they have a value around 0. In Figure [4.1] we can see how these two might look like. The column names are intended to show correlation with regards to the first value ID as pos. corr, correlates positively with ID. From this figure we can also see the necessity to not only look at the correlation coefficients but also the scatter plots as the algorithm tries to find a correlation which might or might not exist. We can see this even better when we look at Ancombes quartet [Ans73] in Figure [4.2] where without plotting we could assume that all of the datasets are the same as the others as they all have the same mean, variance and Pearson correlation coefficient. However through plotting them we can see that they

are completely different in their values and for dataset 2 we can additionally see that a linear regression is completely wrong for that case.

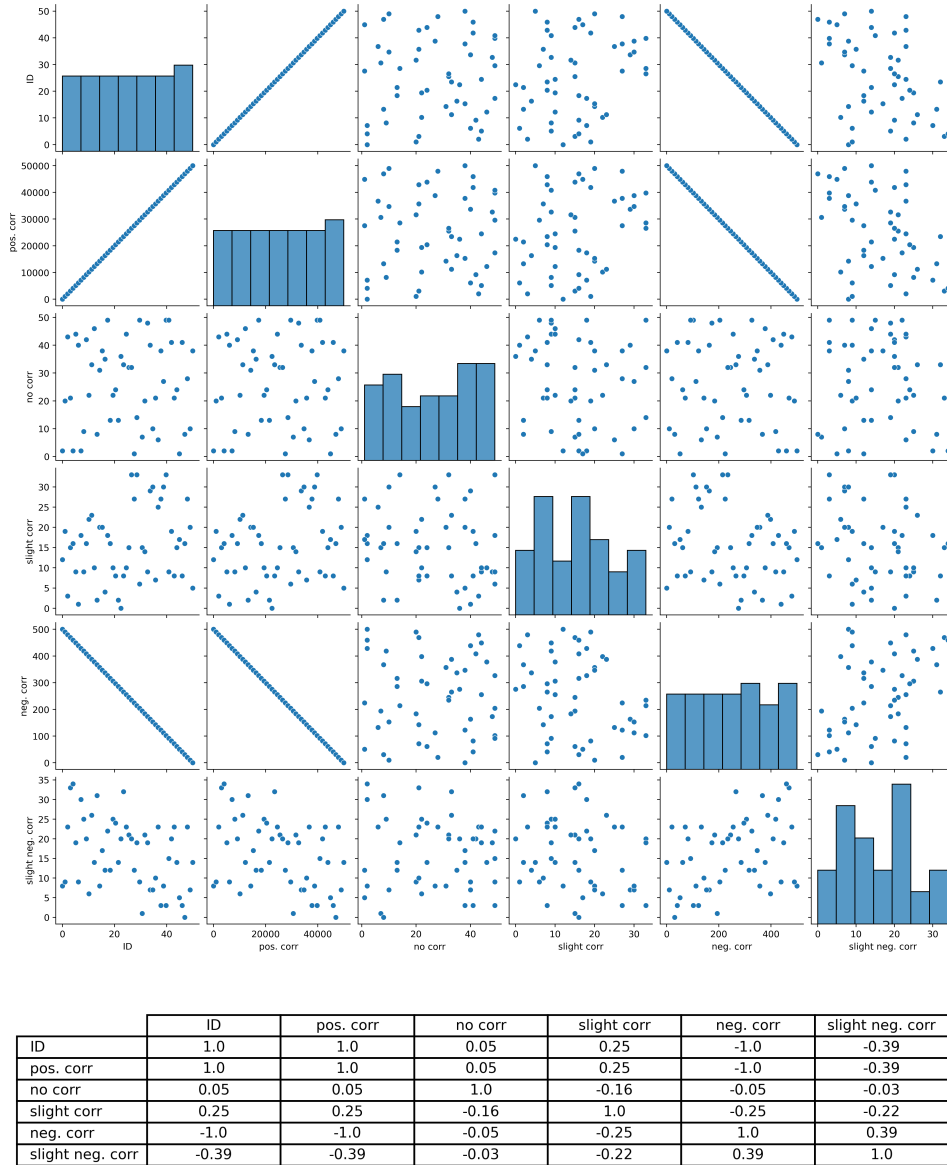


Figure 4.1: A seaborn pairplot with the different parameters and how they correlate with another as well as their histogram. Additionally below is the correlation matrix with its coefficients.

Anscombe's Quartet

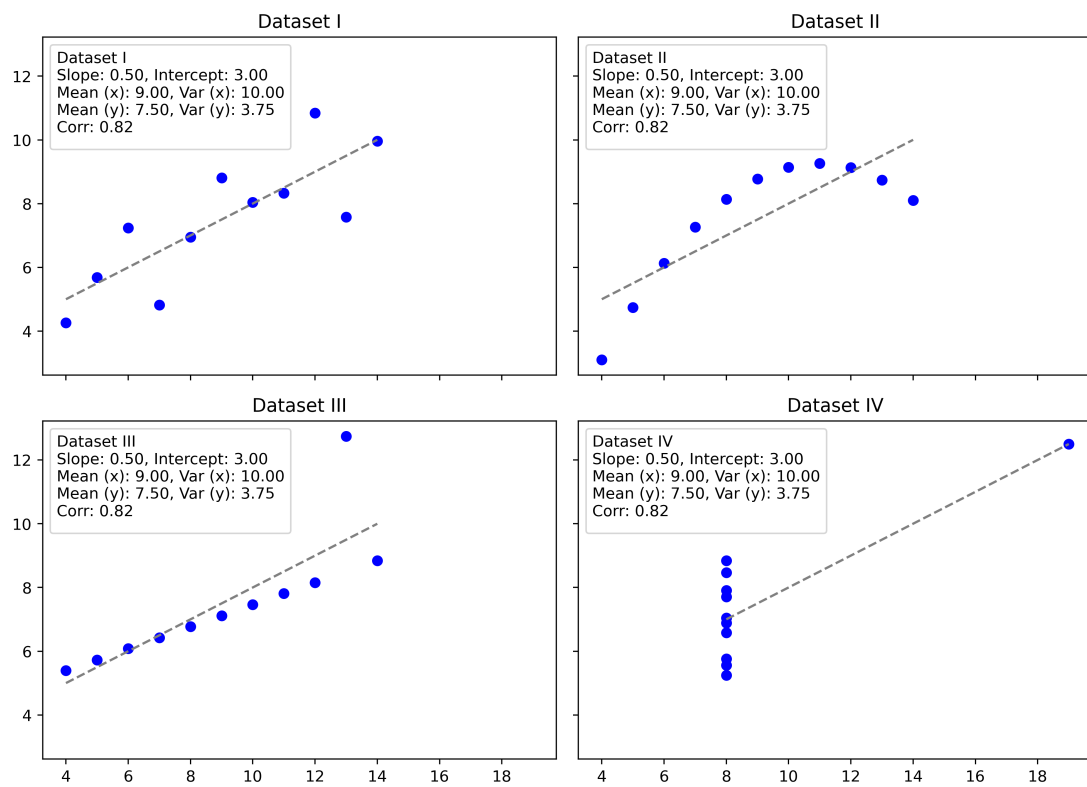


Figure 4.2: Anconbes quartet where four different datasets have the same mean, variance and Pearson correlation values. Ans73

4.2 Preprocessing

We learned that for Gradient Descent to work we have to **scale** the different features. [ES24] We do this via the Python package scikit-learn standardscaler which standardises features by subtracting the mean and scaling to unit variance. However this cannot be applied to all the above mentioned features that we will initially use. The relevant feature we are talking about is the ϕ feature which has a range of 0 to 2π . The standardscaler would classify 0 as -1 and 2π as 1, however since it is a circle 2π should be -1 too. To circumvent this we transform ϕ and θ into x, y and z coordinates through standard conversion between spherical and cartesian coordinates:

$$x = \sin \theta \cos \phi \quad (4.1)$$

$$y = \sin \theta \sin \phi \quad (4.2)$$

$$z = \cos \theta \quad (4.3)$$

The radius in these formulae is one as the misalignment angles depict a unit vector. Additionally we thought about reducing features via substitution. For this we converted the cartesian coordinates of the grains into only the radius which gives us a measure of distance from the centre:

$$r = \sqrt{x^2 + y^2 + z^2} \quad (4.4)$$

4.3 From data to graphs

So now we got the data, next we want to transform it into graph form. The jraph package [jra] has a set form for its graph called GraphsTuple which consist of seven different values:

- nodes - array of node features per grain
- edges - array of edge features per edge
- globals - array of global features
- senders - array of nodes from which the connection to another node commences
- receivers - array of nodes which receive a connection from another node
- n_node - number of nodes
- n_edge - number of edges

It is of importance to use jax [jax] arrays for future training and evaluation steps. The transformation from the initial .statpoly, .krn and .dat files starts by getting the number of neighbours each grain has and their ids. From this we can build the list of outgoing and incoming connections e.g., [1,1,1] and [2, 3, 4]. In this example we have 4 nodes,

4 Data and graph representation

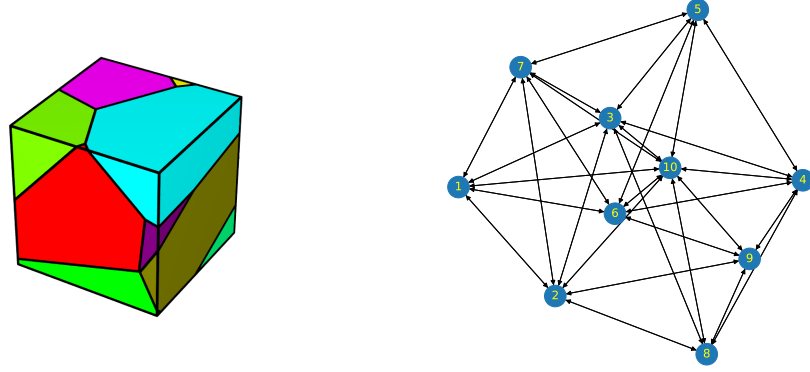


Figure 4.3: Here we see how a magnet with 10 grains on the left would look like in graph form.

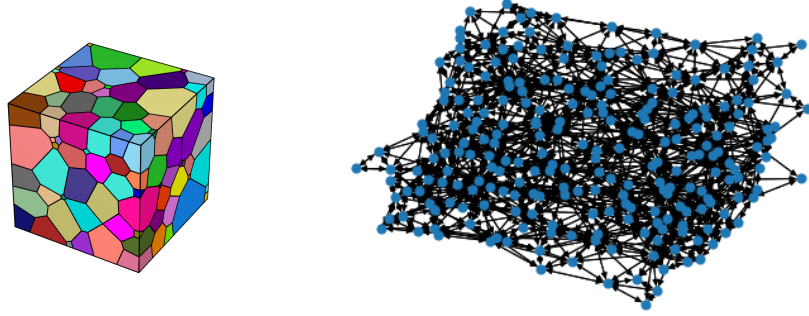


Figure 4.4: How a magnet with 243 grains would look like in graph form.

where from node 1 all directed connections lead to the other nodes. For the connection to be undirected we would have to include the connections back too with $[1,1,1,2,3,4]$ and $[2,3,4,1,1,1]$. Next all other features from the .statpoly and .krn files are collected in the array nodes and the coercivity from hc.dat is collected in globals. As there are currently no edge features the array edges is left empty. Finally the number of nodes and edges is obtained via checking the size of the nodes and edges arrays respectively.

So how would this look like in concrete examples? Lets look at a simulation of a magnet with 10 grains and its corresponding graph form in Figure 4.3.

As we can see the magnet has 10 grains and their neighbours are connected shown in graph form via undirected arrows. With 10 grains it is still easy to see some kind of pattern or rather form taking place when the magnet is in graph form. However this changes quickly when the amount of grains is increased to 243 in Figure 4.4.

The system size is still only 243 grains but the graph form is so convoluted that we cannot see any patterns, but neural networks can.

5 Training and optimization

In this chapter we will look at different datasets and what they are made of by analysing their features through a correlation matrix. Then training is done on them via different graph neural networks and the dataset as well as the graph neural network will be optimized according to the mentioned benchmarks for classification. Additionally we will look at different parameters like the importance of dataset size and variation.

5.1 Neural network architecture

First and foremost we should talk about the package used to implement the graph neural network. The graphs structure and message passing is done via the package jraph [jra] a python package that utilises the package jax [jax] for computation. The neural networks are generated through the package haiku [dmh] and every node, edge and global update uses a different neural network. For edge and global updates the neural networks are rather simple and can be seen in Table 5.1. The node update neural networks in turn look exactly the same as the edge update neural network with the slight difference of a variable depth that specifies the amount of haiku.linear layers before the last haiku.linear layer. Meaning that a depth of one is equal to the neural network shown in the edge update neural network example. The optimizer used was initially adam but later changed to adamw both from the optax package as it has built in weight decay. Finally the predictions were done through a logarithmic softmax and subsequent cross entropy loss function, which is used because we model the binary classification task through two neurons. This not only gives us an insight into the probabilities of the two classes but the logarithmic softmax function also prevents numerical instability.

layer type	shape	layer type	shape
haiku.linear	number of neurons	haiku.linear	number of neurons
jax.nn.relu	number of neurons	jax.nn.relu	number of neurons
haiku.linear	number of neurons	haiku.linear	2

Table 5.1: A general overview of the neural network used for edge updates on the left and global updates on the right.

5.2 Dataset 1: Low coercive grain

The first dataset we will look at will be more of a proof of concept as there is no variation of the grain structure across the dataset.

5 Training and optimization

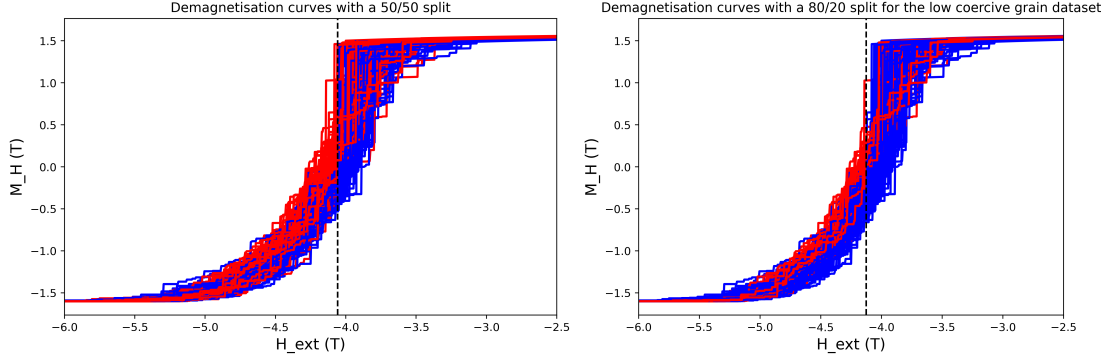


Figure 5.1: Demagnetisation curves split into 50/50 and 80/20 for the low coercive grain dataset

5.2.1 Dataset description

This dataset consists of 157 graphs with the only difference between them being a shuffled misalignment angle. In particular the θ of the misalignment angles is shuffled between 0 and 15 degrees with an additional grain where its θ is set to 45 degrees. The full parameters of the dataset can be seen in Table 5.2

Graphs in Dataset	Nr. Grains	Nr. magnetic Grains	Nr. Edges	H_c range (T)
157	243	157	2472	3.87 - 4.26

Table 5.2: Parameters for the first dataset.

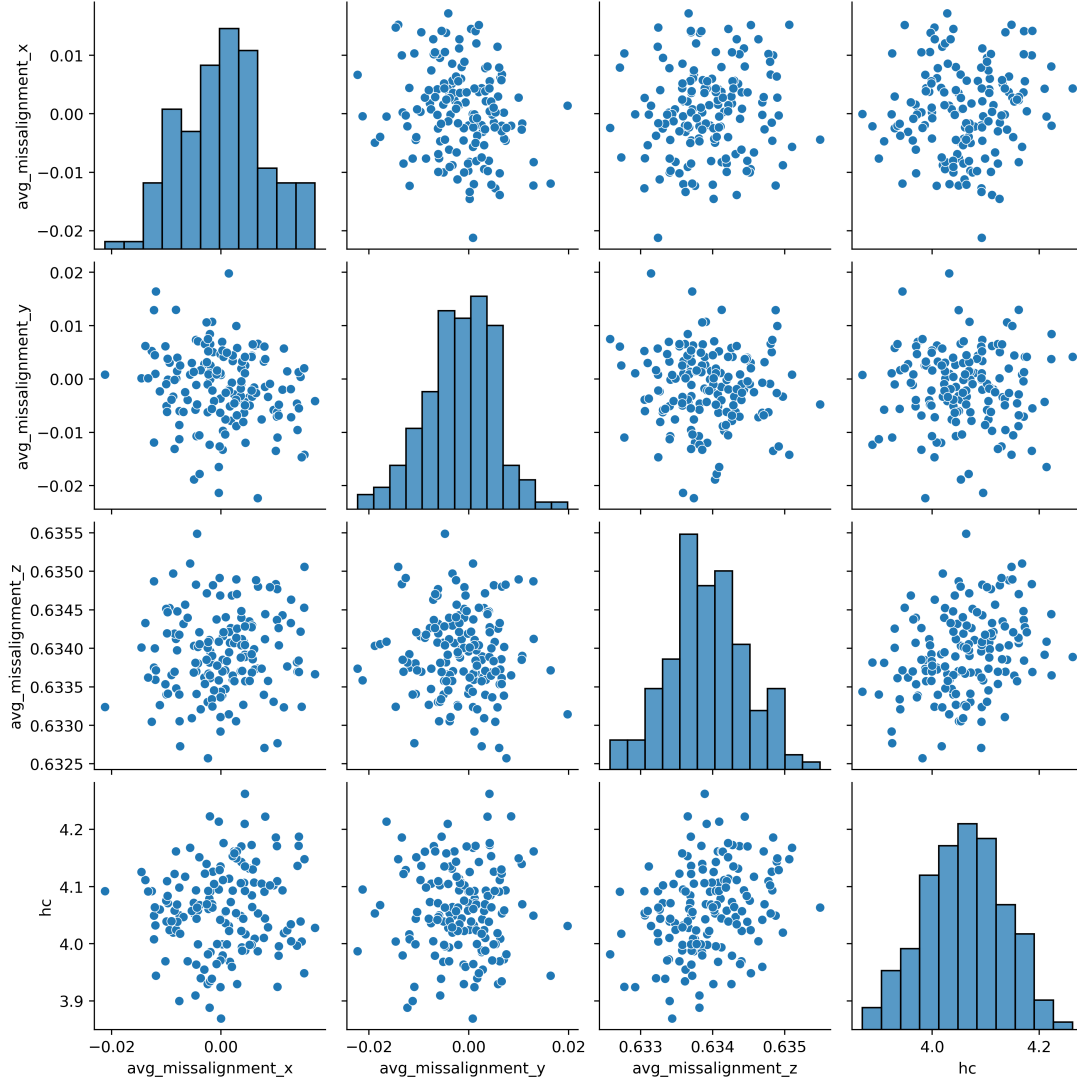
One can say that the number of edges should be halved as the number here represents the number of edges needed for the graph package to build a undirected graph. This means that we have two edges between the same nodes, one outgoing and one incoming. For all coming datasets and graph descriptions this will be the case too. Finally, as we will split the dataset according to its coercive field we can see how these splits will look like when plotting the hysteresis curves in Figure 5.1. In that figure we can see that the curves are bundled together and overlap even in some parts.

5.2.2 Feature analysis

We have mentioned it in the prior section, but the only variation in this dataset stems from the variation of the misalignment angle. Lets look at the correlation matrix of a graph and at the correlation matrix across the whole dataset with the features averaged over one graph to represent each graph as one sample.

First we look at the variation across the whole dataset in Figure 5.2. As we heard the only variation is in the misalignment angle. Therefore we will only look at the correlation between them and the coercivity. From the histogram plots we can see a normal distribution for all the variables and from the scatter plot we cannot clearly see

5.2 Dataset 1: Low coercive grain



	$avg_missalignment_x$	$avg_missalignment_y$	$avg_missalignment_z$	hc
$avg_missalignment_x$	1.0	-0.17796	0.09232	0.08577
$avg_missalignment_y$	-0.17796	1.0	-0.05126	-0.00885
$avg_missalignment_z$	0.09232	-0.05126	1.0	0.29851
hc	0.08577	-0.00885	0.29851	1.0

Figure 5.2: Correlation matrix as well as plot of the whole badgrain dataset.

5 Training and optimization

any correlation except for maybe the misalignment_z and coercivity variables. When we look at the table we can again see that the most correlation between the variables is between the misalignment_z and the coercivity. Which is quite logical as the worse the average z coordinate in a graph is the worse the overall coercivity will be. With worse being a smaller z value as this means a high θ and therefore a worse overall alignment of the grains.

Now lets look at the correlation between features in one graph in Figure 5.3. For analysing purposes we left all features in, however when we train with the dataset we will only train with the misalignment angles. The last few rows of the matrix as well as of the scatterplot are not meaningful since there is no variation even in the graphs. This is because the anisotropy, the magnetic saturation as well as the exchange constant is for every magnetic and non magnetic grain the same. Also with the misalignment angles we see the same trend as all non magnetic grains simply have zeroes for the magnetic properties. From this we would argue that any analysis for at least the anisotropy, the magnetic saturation as well as the exchange constant is non conclusive. In following correlation plots these will be left out. Now lets look at each column, beginning with the radius. We can see negative correlation between radius and dihedral angles as well as between radius and sphericity. For sphericity this stems from the fact that grains on the edge of the system especially the corners are modelled as corners removing them from a spherical form. The same reasoning can be applied to the two angles. Next in the minimal dihedral angle column we can see a correlation with sphericity as the bigger the minimal angle the more the grain resembles a sphere. The same reasoning can be applied to the correlation with the equal diameter and the maximal dihedral angle. The bigger the minimal dihedral angle the more spherical the grain is, therefore occupying more space. For the maximum dihedral angle we apply the same reasoning, but we can also see a correlation between it and the misalignment in z direction which is quite interesting. But when we look at the corresponding scatter plot we can see that there is no real correlation visible. Next for sphericity we have a relative high correlation with equal diameter which stems from the fact that if a grain is spherical it occupies more space and has therefore a bigger diameter. Finally for the histograms we mainly see a normal distribution with outliers beginning with the equal diameter histogram. Here the first bar stems from the non magnetic grains that are in comparison smaller than the magnetic ones. For the x misalignment we have a prominent centre bar as all non magnetic grains are zero. Which is the same for the y and z misalignment angles as well as the last three magnetic properties. Lastly for the one outlier of the y and z misalignment angle which stems from the one grain that has a θ of 45 degrees. As no variation in features is present only the misalignment angles shall be chosen as significant features for the training process. It was also tested if the position of the worst grain inside of the magnet effects the coercivity. However no correlation was found.

5.2.3 Training and optimization

We start by splitting the dataset into 50% bad and 50% good magnets according to their coercivity. Meaning that the worst 50% of magnets will get a target value of 0 and

5.2 Dataset 1: Low coercive grain

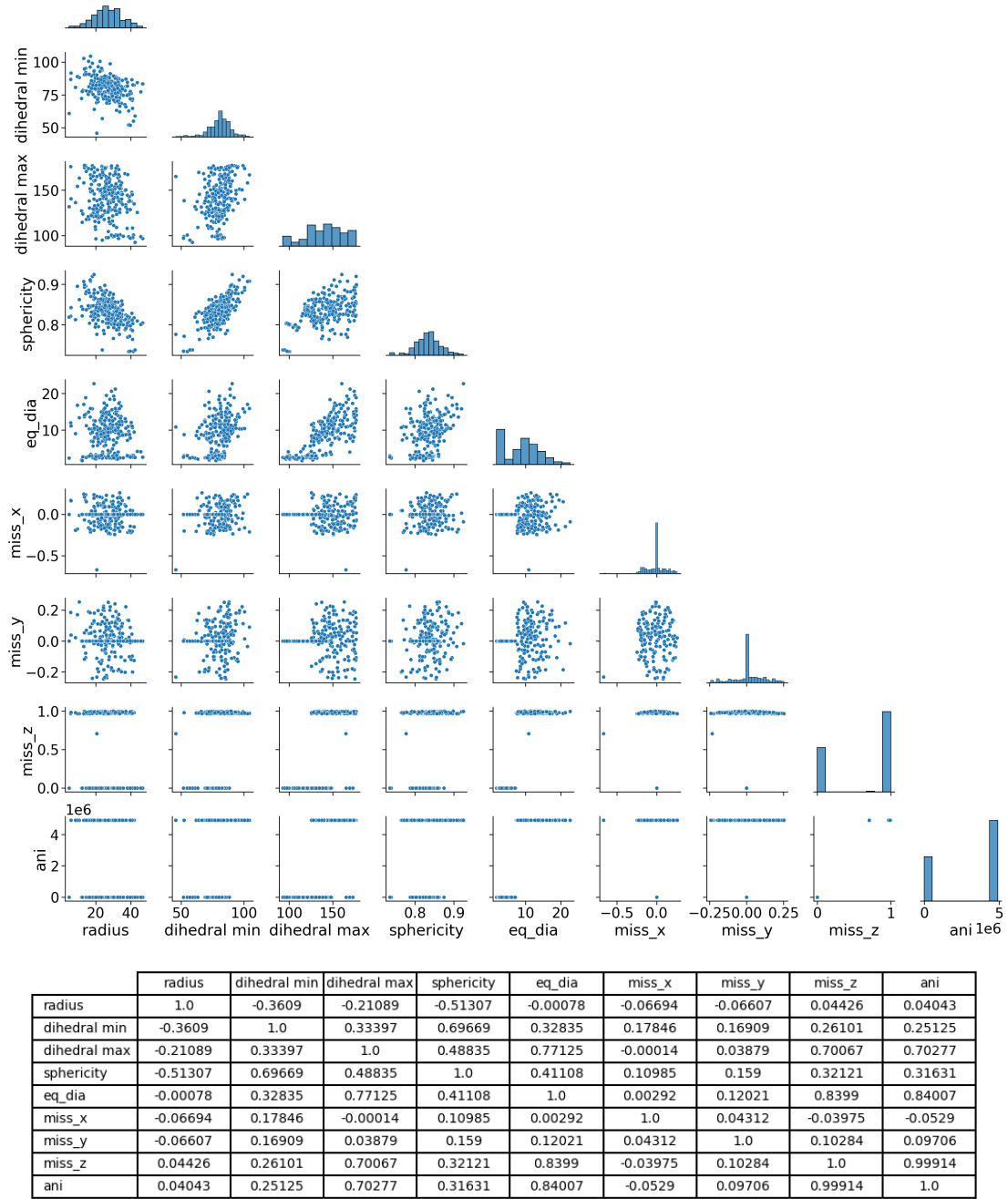


Figure 5.3: Correlation matrix as well as plot features of one graph in the bad grain dataset.

5 Training and optimization

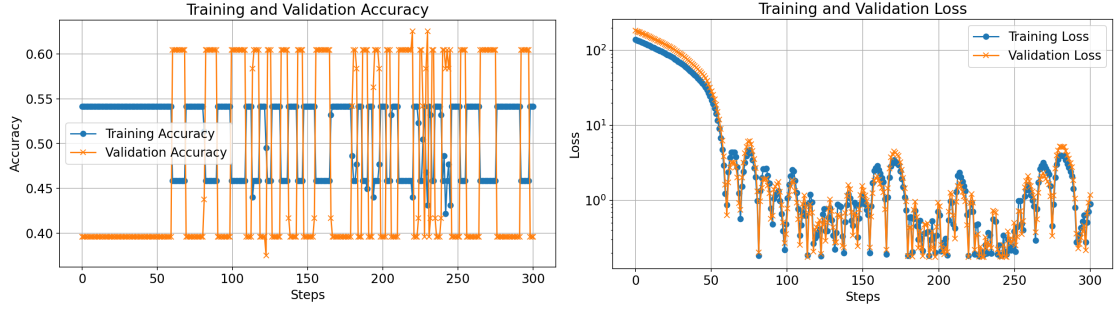


Figure 5.4: Plot of the loss and the accuracy curves for the first network without optimization.

the other 50% a target value of 1. The 50% threshold is chosen to ensure homogeneity between the two classes. After that we split the dataset into the Training and Validation datasets by 66% and 33% and run a standard-scaler from scikit-learn [sci] on the training dataset. This ensures that the network does not learn anything about the other dataset through its scaling.

The neural network in its standard form has one hidden layer with 32 neurons summing to 7362 parameters. For the first run a stochastic gradient descent approach was chosen. We can see this first run with its loss and accuracy in Figure 5.4 and its corresponding confusion matrix in Table 5.3.

True/Predicted	Class 0	Class 1
Class 0	21	8
Class 1	10	9

Table 5.3: Confusion Matrix for the validation set

From this initial run we can see that the training accuracy never improved above the trivial solution of classifying everything as one class. In the validation set we can see that it at least tried to classify some correctly according to the confusion matrix. Therefore as the dataset is also rather small it was deemed that the model overfitted. The approach was then changed to minibatch gradient descent as the batch size adjustment to one or the whole dataset opens up the possibility to explore either stochastic gradient descent as well as batch gradient descent from it. Since the continuous gradient and weight updates from stochastic gradient descent have shown no improvement in accuracy or the confusion matrix. Additionally to this change a learning rate scheduler was introduced into the system to dynamically change the learning rate of the neural network during the training process. This in general is done because a higher learning rate at the beginning is to get out of local minima and approach the target faster and a lower learning rate at the end is to hone in on the global minima. Then we included L2 regularization as well as

5.2 Dataset 1: Low coercive grain

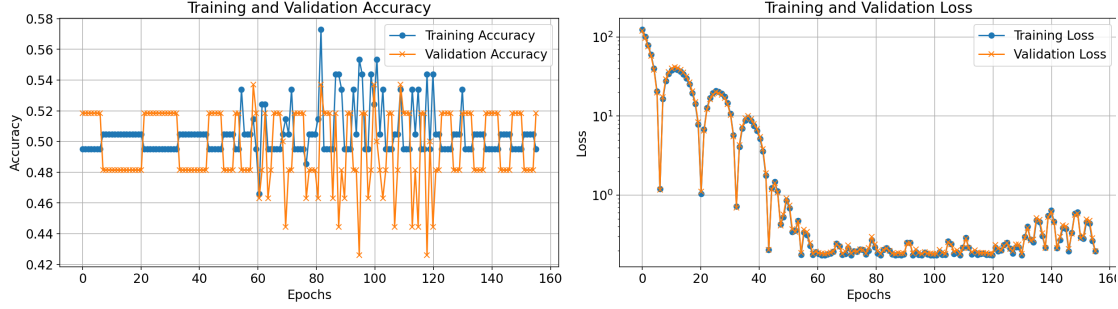


Figure 5.5: Best result for the training set during grid search

a early stopping mechanism. L2 was chosen because the non necessary features were already filtered out beforehand, to ensure that the model is not overly reliant on one specific node feature and to minimize the weights of the network. Lastly a early stopping mechanism was implemented to stop the training when no improvement in validation loss can be measured. Again as a measure to stop overfitting. Next a grid search was done varying the hyperparameters batch size, hidden layer count and neurons per layer. For the learning rate scheduler a starting value of 10^{-3} and a finishing value of 10^{-6} was deemed good. It was implemented in a way that is coupled to the batch size. The weight decay, alpha from eq. 3.12, was chosen to be 10^{-4} and the patience was also coupled to the batch size since a smaller batch size is expected to converge slower to the ideal solution and therefore needs a higher patience. For the two values batch size and neurons multiples of 2 were chosen for the grid search.

From the grid search we again see two scenarios, either only the trivial solution was chosen by the model or only one of the two sets used during training achieves an accuracy above the trivial solution. In Figure 5.5 we see the best result for the training set which also achieved accuracy above the trivial solution for the validation set. The exact parameters for this run are a batch size of 8, 16 neurons per layer and only one layer summing to 2146 parameters. The confusion matrix for the validation set is given in Table 5.4.

True/Predicted	Class 0	Class 1
Class 0	21	7
Class 1	18	8

Table 5.4: Confusion matrix for the validation Set

Which might seem worse than for the initial run, however we have to remember that the trivial solution of only classifying the whole set as one class for the initial run was already above 60% accuracy. The accuracy for the training set is more interesting as a +6.7% above the trivial solution was achieved. For the same model parameters another

5 Training and optimization

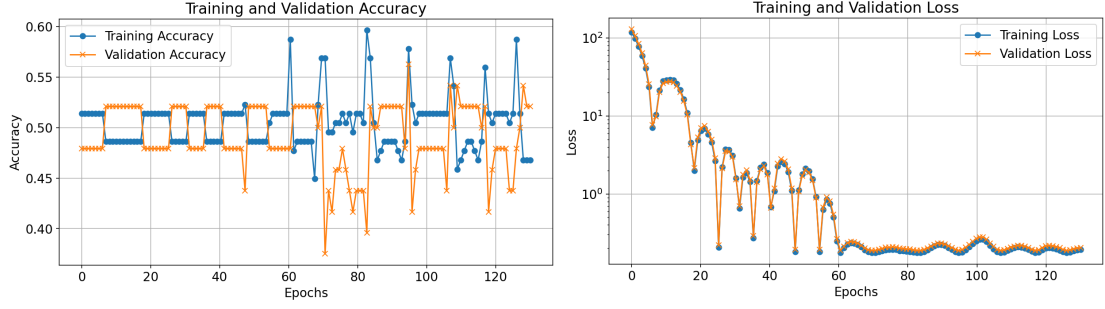


Figure 5.6: Accuracy and loss after optimization with weight decay 10^{-4}

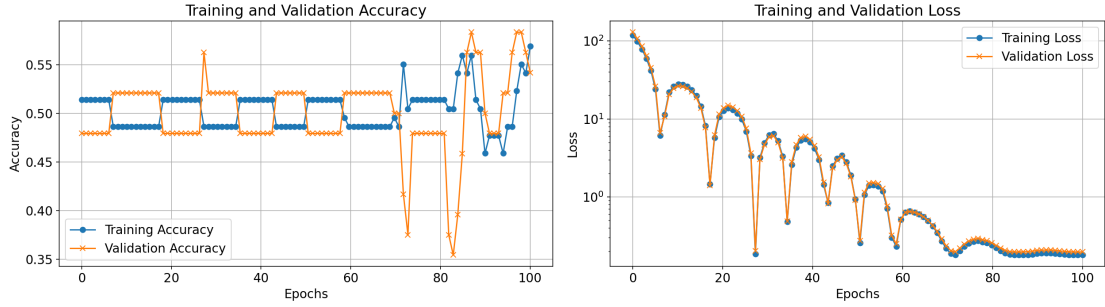


Figure 5.7: Accuracy and loss with weight decay = 10^{-5}

grid search was started to solidify the chosen hyperparameters of the learning scheduler and the weight decay as well as to see if an improvement in the validation set could be achieved. It was found that the prior chosen hyperparameters were nearly chosen correctly with only minor adjustments to the learning rate scheduler to a range of 10^{-3} to 10^{-5} and the weight decay was set to 10^{-4} and 10^{-5} . The accuracy and loss function for a weight decay of 10^{-4} can be seen in Figure 5.6 and for a weight decay of 10^{-5} in Figure 5.7 and their corresponding confusion matrices can be seen in Table 5.5

Although we could not improve upon the accuracy of the sets we could however narrow the accuracy between them. Whereas we had a difference in accuracy between the sets of around 4% prior to weight decay and learning rate adjustment we now have a difference of around 1% in the cases highlighted above. Lastly we wanted to see if we could classify

True/Predicted	Class 0	Class 1	True/Predicted	Class 0	Class 1
Class 0	10	13	Class 0	8	15
Class 1	8	17	Class 1	5	20

Table 5.5: Confusion matrix for a weight decay 10^{-4} on the left and for a weight decay of 10^{-5} on the right

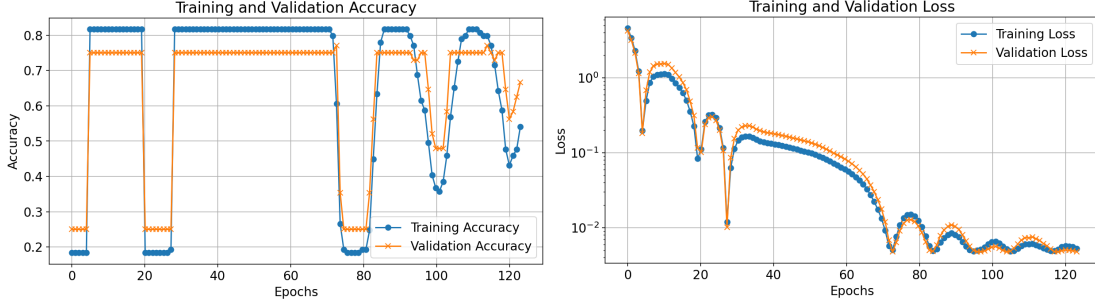


Figure 5.8: Accuracy and loss for 80/20 split dataset

only the best 20% by splitting the coercivity into 80% worst and 20% best magnets from now on called 80/20 split. This was done as the coercivity is normal distributed which makes it harder for the classifier to find differences in the magnets with coercivity around the mean value of 4 T. As this introduced class heterogeneity we had to adjust our loss function to penalize the model harder for wrongly classifying good magnets. This is simply done via introducing a weight into the loss function through inverse frequency weights utilised from [HG09]. The results can be seen in Figure 5.8 and its corresponding confusion matrix in Table 5.6.

True/Predicted	Class 0	Class 1
Class 0	28	8
Class 1	3	9

Table 5.6: Confusion Matrix for the 80/20 split dataset

From Figure 5.8 we can see that an accuracy above the trivial solution for the validation set could be achieved however in the same step at 118 the training accuracy went below it. It is also nice to see that the wrongly classified good magnets are one third of all the good magnets. Naturally we would want this number to be even lower. The hyperparameters were again varied via grid search for this run with the following parameters for the shown run: batch size 8, neurons per layer 32, depth 2 and weight decay of 10^{-5} summing to 8418 parameters.

5.3 Dataset 2: Grain size

5.3.1 Dataset description

This dataset is comparatively even smaller than the previous one with only 50 graphs. However it consists of 11 different structures in five different magnet sizes. Some of the calculations failed that is why there are not 55 graphs. The specific values can be seen in Table 5.7 and the split demagnetisation curves can be seen in Figure 5.9. In contrast to

5 Training and optimization

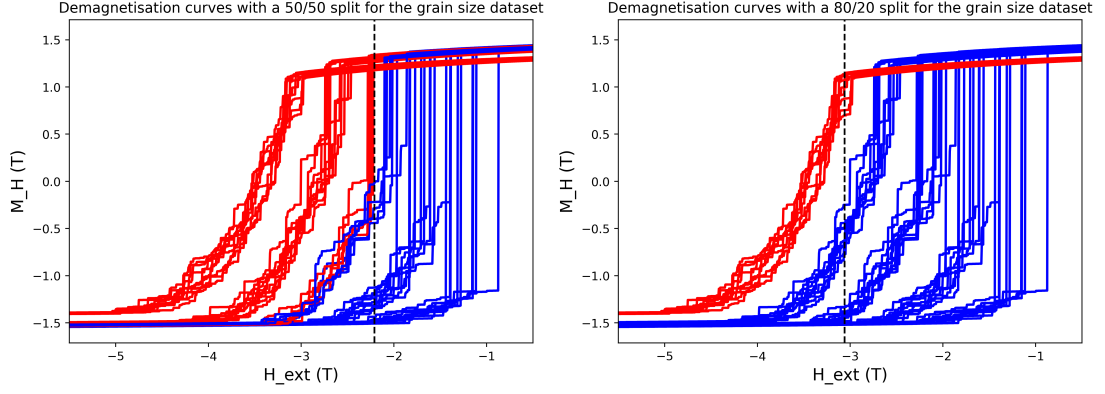


Figure 5.9: Demagnetisation curves for the grain size dataset split in either 50/50 or 80/20

the first dataset these demagnetisation curves clearly are separated, which might help the neural network distinguish them.

Graphs in Dataset	Nr. Grains	Range magnetic grains	Range edges	H_c range (T)
50	243	155 - 158	2464 - 2508	0.875 - 3.53

Table 5.7: Parameters for the grain size dataset

5.3.2 Feature analysis

As previously mentioned we have different magnet sizes, but not an increased amount of grains. This indicates a variation in grain size between the different magnets. Therefore we will look at the average radius across the dataset to see any correlation with the coercivity. Alternatively the same could have been done with the equal diameter. The result of the correlation can be seen in Figure 5.10 and from it we can deduce that the coercivity decreases with magnet size. The correlation of the features inside of the graphs is not shown again as the results were similar to the analysis from the previous dataset.

5.3.3 Training and optimization

The same approach as with the previous dataset was chosen and a grid search for the different parameters was done, varying the neurons per layer, the depth, the batch size and the weight decay. Again no test set was made as the size of the dataset is too small to include it. For the first run the split for the coercivity was chosen to be 50/50. It was found that a batch size of 16, a depth of 2, neurons per layer 16 and a weight decay of 10^{-4} was the least complex neural network that achieved 100% validation accuracy with the result shown in Figure 5.11 and corresponding confusion matrix in 5.8. For this network layout, the number of unknowns to be trained is 2418 parameters. To elaborate

5.3 Dataset 2: Grain size

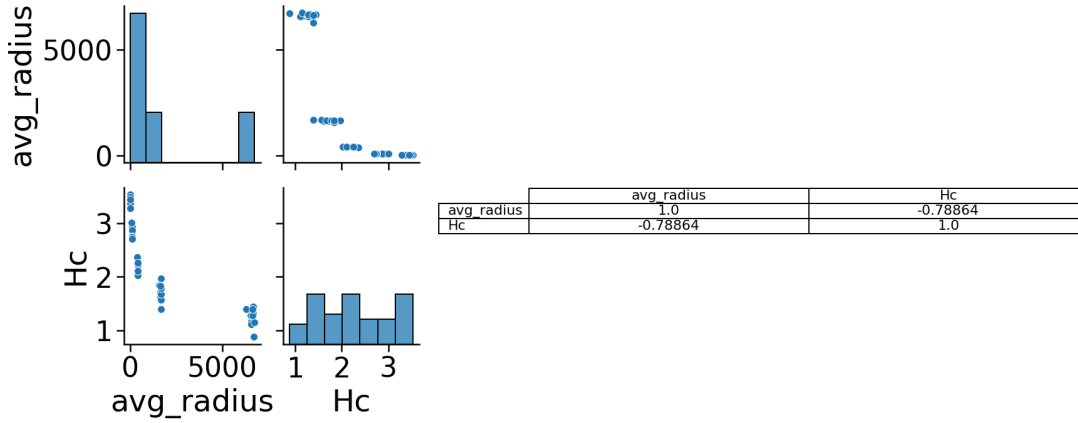


Figure 5.10: Correlation matrix of the grain size dataset with only radius and coercivity shown.

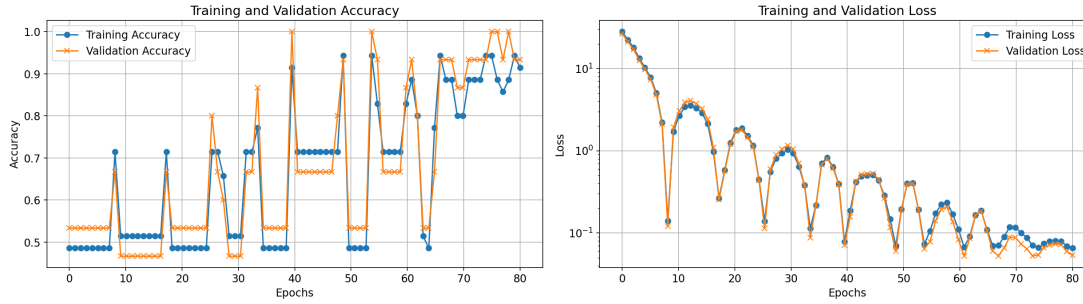


Figure 5.11: Accuracy and loss for a 50/50 split grain size dataset

a few other hyperparameter compositions also achieved the same accuracy scores for the training and validation set however in that case computational efficiency was chosen for the result parameters listed. For example a deeper neural network would need more computational resources and would also take longer during training.

True/Predicted	Class 0	Class 1
Class 0	7	0
Class 1	0	8

Table 5.8: Confusion Matrix for the 50/50 split validation dataset

For the second test the datasets coercivity was split 80/20 and another grid search was started but with values around the previously found ones. The result can be seen in Figure 5.12 and Table 5.9. The hyperparameters stayed the same except neurons per layer 32 and a weight decay of 10^{-5} summing to 8418 parameters. The highest achieved

5 Training and optimization

validation accuracy is above 90% with all the good magnets correctly classified.

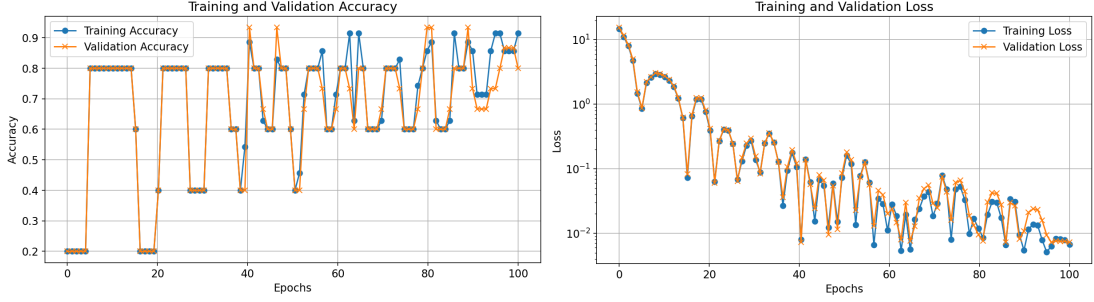


Figure 5.12: Accuracy and loss for the 80/20 split 2. dataset

True/Predicted	Class 0	Class 1
Class 0	11	1
Class 1	0	3

Table 5.9: Confusion Matrix for the 80/20 split validation dataset

5.4 Dataset 3: Secondary phases

5.4.1 Dataset description

This data set contains microstructures with magnetic and non-magnetic grains. Non-magnetic grains form at corners of magnetic grains and are often called triple junctions. We have 12 granular structures with varying appearances in the dataset for a whole 366 magnets. For each structure we have different easy axes distributions and different occurrence of oxides. Oxides are a new global features which indicate that grain surfaces are covered with oxides that reduce the Stoner-Wohlfarth switching field at the respective evaluation point in the reduced order model. Other parameters of the dataset can be seen in Table 5.10 and the split demagnetisation curves in Figure 5.13. We see a new variation of demagnetisation curves, that are again bundled together. However different to the first dataset we have clear lines that do not overlap most of the time.

Graphs in Dataset	Range grains	Magnetic grains	Range edges	H_c range (T)
366	650 - 3250	650	7370 - 37750	0.855 - 2.615

Table 5.10: Parameters for the abnormalities data set.

5.4 Dataset 3: Secondary phases

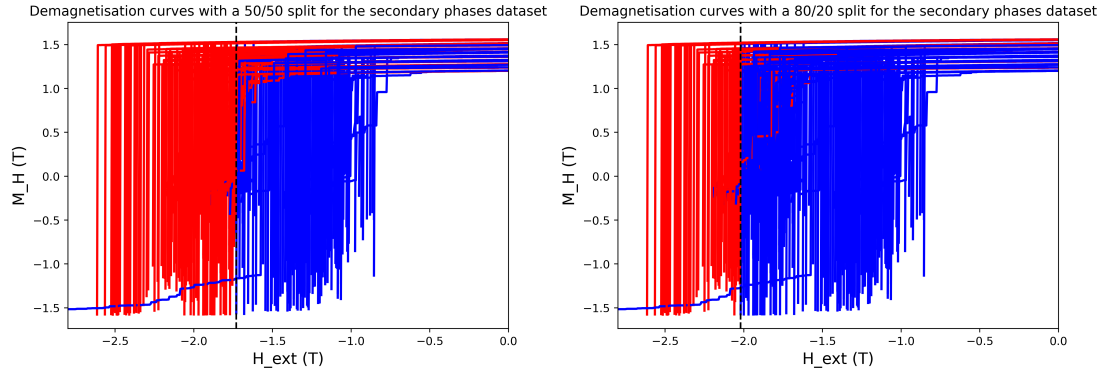


Figure 5.13: Demagnetisation curves for the secondary phases dataset. Not the full curves can be seen but only when they switch magnetisation and stopping there.

5.4.2 Feature analysis

As there are new features implemented in this dataset we are mainly interested in their behaviour and correlation with the coercivity. A correlation matrix with the most interesting features was made and can be seen in Figure 5.14. Unfortunately as the triple junctions and oxide can only be there or not we cannot see a clear relation from the scatter plots. However it seems like the coercivity decreases slightly when oxide are present. Also we can see that the distribution of the coercivity is not really normal distributed. Finally the other features, such as sphericity, are not looked at as there is nothing new to learn in comparison to the prior datasets.

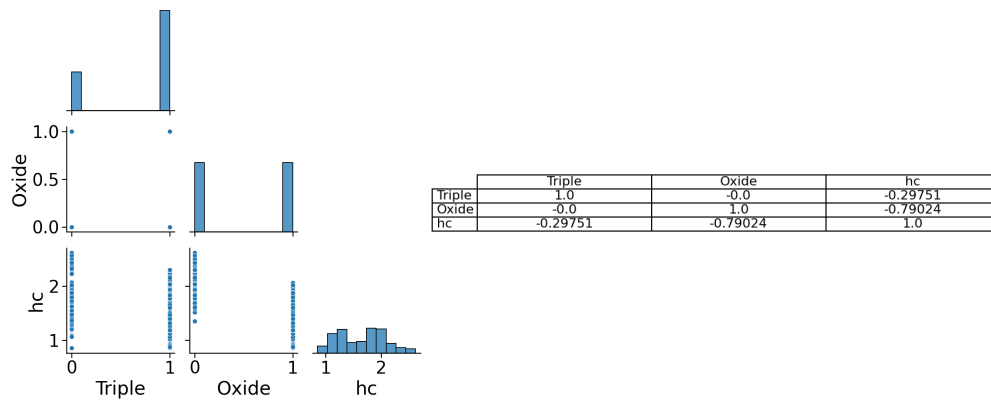


Figure 5.14: Correlation matrix of the abnormalities dataset with only the newly introduced features

5 Training and optimization

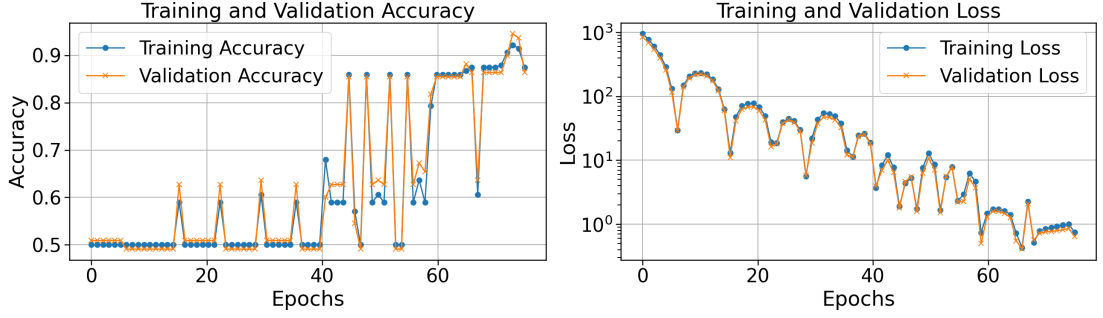


Figure 5.15: Accuracy and loss for the 50/50 split validation set.

5.4.3 Training and optimization

The same grid search as for the previous datasets was repeated first for a coercivity split of 50/50. It was interesting to see that the previously found best hyperparameters seemed to repeat. For the best accuracy of 94% in the validation and 92% in the training set we had 16 neurons per layer, a batch size of 16, one hidden layer and 10^{-5} as weight decay summing to 2146 parameters. The complete loss and accuracy per epoch can be seen in Figure 5.15 and the corresponding confusion matrix in Table 5.11 on the left. The best run was not only chosen by its accuracy but also by its recall score as no good magnet was wrongly classified. We can also see that in comparison to the previous dataset the accuracy does not fluctuate as much at the end. What is interesting though is that the smallest loss is not at the best accuracy which seems counter-intuitive.

True/Predicted	Class 0	Class 1	True/Predicted	Class 0	Class 1
Class 0	50	6	Class 0	78	12
Class 1	0	54	Class 1	2	18

Table 5.11: Confusion Matrix for the 50/50 split validation dataset on the left and 80/20 split on the right.

Next we tried the 80/20 split again as this would fall seemingly in the second peak of the coercivity histogram. From this we could argue that the increased amount of data around the 80% point could give us an insight in how accurate the model really predicts the coercivity of each graph. The best validation accuracy was achieved with a batch size of 16, 16 neurons, a depth of 3 and a weight decay of 10^{-5} summing to 2690 parameters. Although an improvement of the accuracy above the trivial solution could be achieved, with 87%, it was not as high as for the 50/50 split dataset. The recall however is very good with only two wrongly classified good magnets.

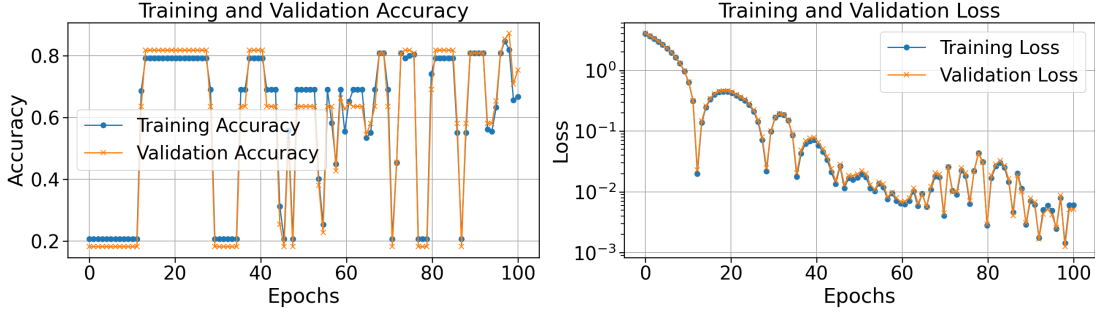


Figure 5.16: Accuracy and loss for the 80/20 split validation dataset.

5.5 Dataset 4: Generalisation

What we want to achieve with neural networks is not only a good accuracy for the seen data but also a good accuracy for the unseen data which is different from that in the training set. In the previous datasets we always had cases where the neural network somehow learns something about the validation dataset, although not through the standard scaler but through other means. In the first set there was no differentiation in structure so the neural network already knew the structures of the validation set. In the second and third dataset the same thing could be argued. Additionally graph neural networks are expected to generalise very well to previously unseen data even for bigger magnet sizes as seen in [SGGP⁺20]. Therefore the next goal would be to show generalisation with a dataset that explicitly only changes the magnet size but no other parameters of the graphs.

5.5.1 Dataset description

This dataset has no repeating structures as each and every simulated magnet is different. Additionally the magnet size slowly increases giving the neural network an idea of how coercivity may change with it. The dataset was split into two right from the beginning, namely into the training part, which will be additionally split into training and validation set, and the generalisation part, which consists only of the biggest magnet sizes. This time there are no non magnetic grains included in the magnets. The general parameters for the two sets can be seen in Table 5.12 with the training and validation set on top and the generalisation set on bottom. We can see that the generalisation set begins with a magnet size similar to the training and validation set and increases to 10x the amount of grains. Also we can see that the coercivity slightly overlaps between the sets. From this we can already know that if we split the training set 50/50 by coercivity that all the graphs in the generalisation set will be bad magnets as the coercivity which splits the good and bad magnets is 3,905 T. Therefore a 20/80 split run may be considered to actually have more than one class in the generalisation set. In Figure 5.17 we can see how the split demagnetisation curves will look like when the dataset is split into two classes. Here the whole dataset is shown including the generalisation set. Similarly to the secondary phases dataset we have bundled curves but no overlapping. At least when the

5 Training and optimization

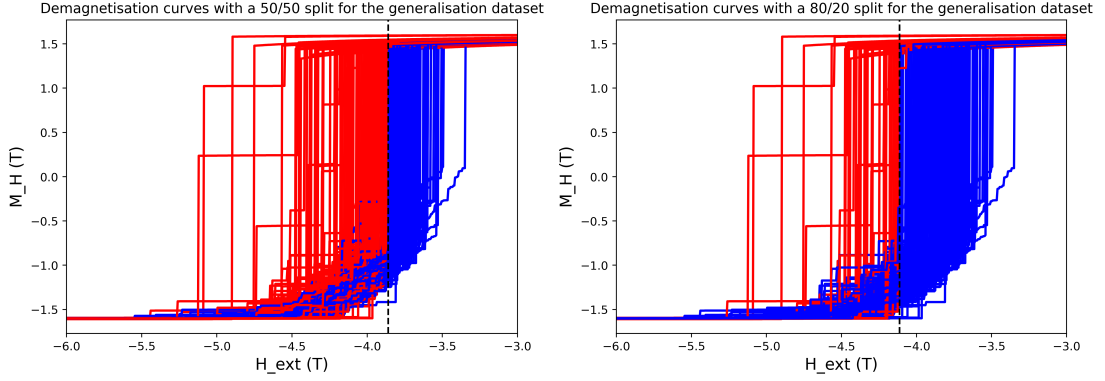


Figure 5.17: Demagnetisation curves for the generalisation dataset split in 50/50 and 80/20.

demagnetisation start till the magnetisation flips signs.

Graphs in Dataset	Range grains	Range edges	H_c range (T)
205	2 - 278	2 - 3008	3.54 - 5.09
33	279 - 3375	3046 - 42556	3.4 - 3.89

Table 5.12: Parameters for the training / validation dataset on top and for the generalisation dataset on bottom.

5.5.2 Feature analysis

We will now look at the features of the two sets. This time not only because every graph is different, but also to check if the generalisation graphs are from the same distribution as the training set. Meaning that the average values are similar to the the trained ones. The finished correlation plot for the whole dataset can be seen in Figure 5.18. From this picture we can see that there exist a high correlation between the average radius as in average distance from the centre of the magnet and the magnet size, which is self-explanatory. We can also see that the magnet size in general correlates with everything to some degree and will therefore be omitted from the dataset during training. It also seems like the average sphericity, the average minimal dihedral angle as well as the average radius correlate with another, which is something we also found with the first dataset. Lastly we see that the coercivity is slightly out of distribution for the larger magnets in the generalisation dataset, because the demagnetising effects increase with the amount of grains, which increase the total magnet size. With increasing magnet size the demagnetization field at the evaluation points near the magnet's surface increases and in turn coercivity decreases. This might pose issues for the neural networks during generalisation. Then in Figure 5.19 we looked at the correlation of the features inside of an graph. In this specific case we chose the biggest graph. We can again see that

5.5 Dataset 4: Generalisation

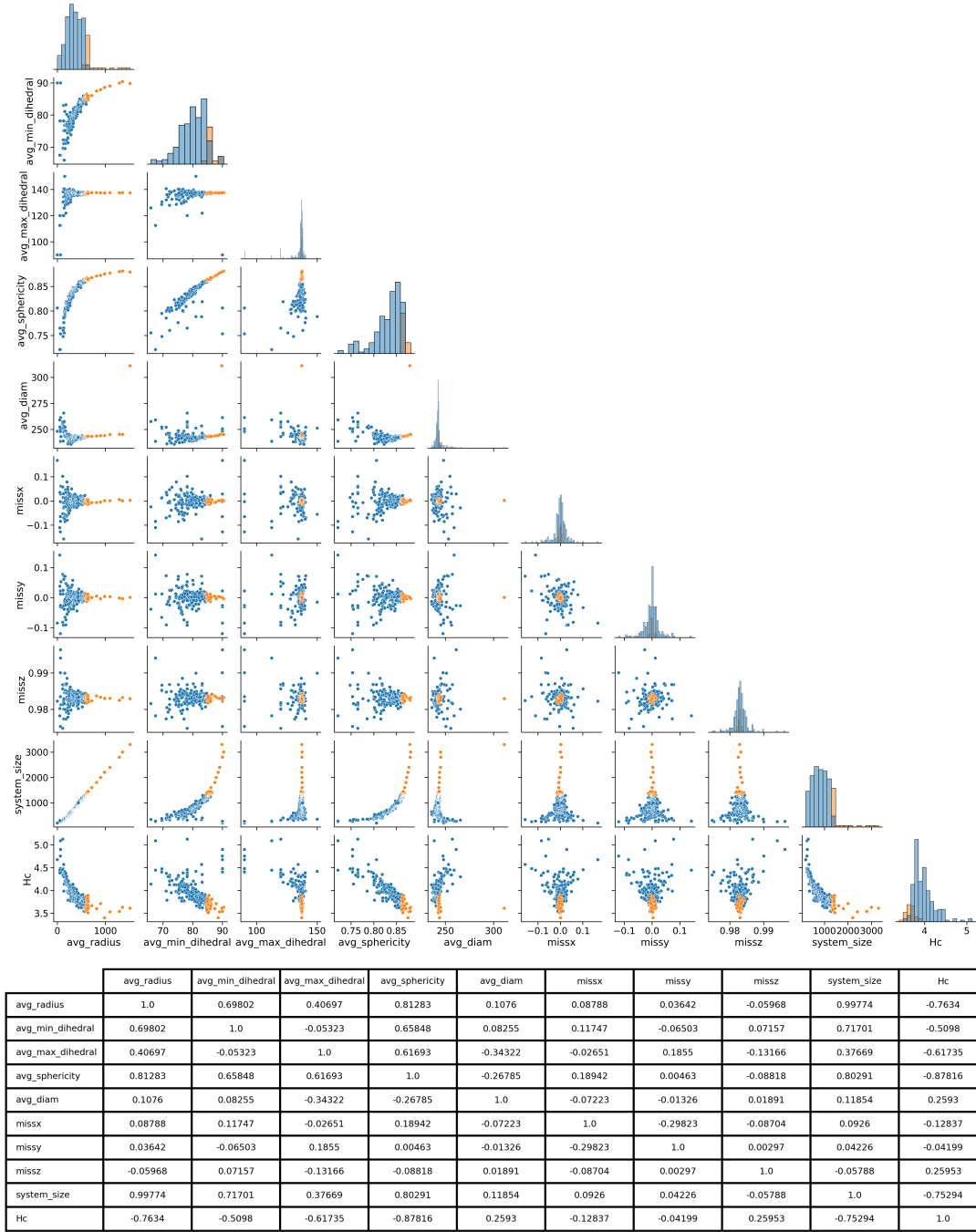


Figure 5.18: Correlation plot for the the training/validation (blue) and generalisation set (orange).

5 Training and optimization

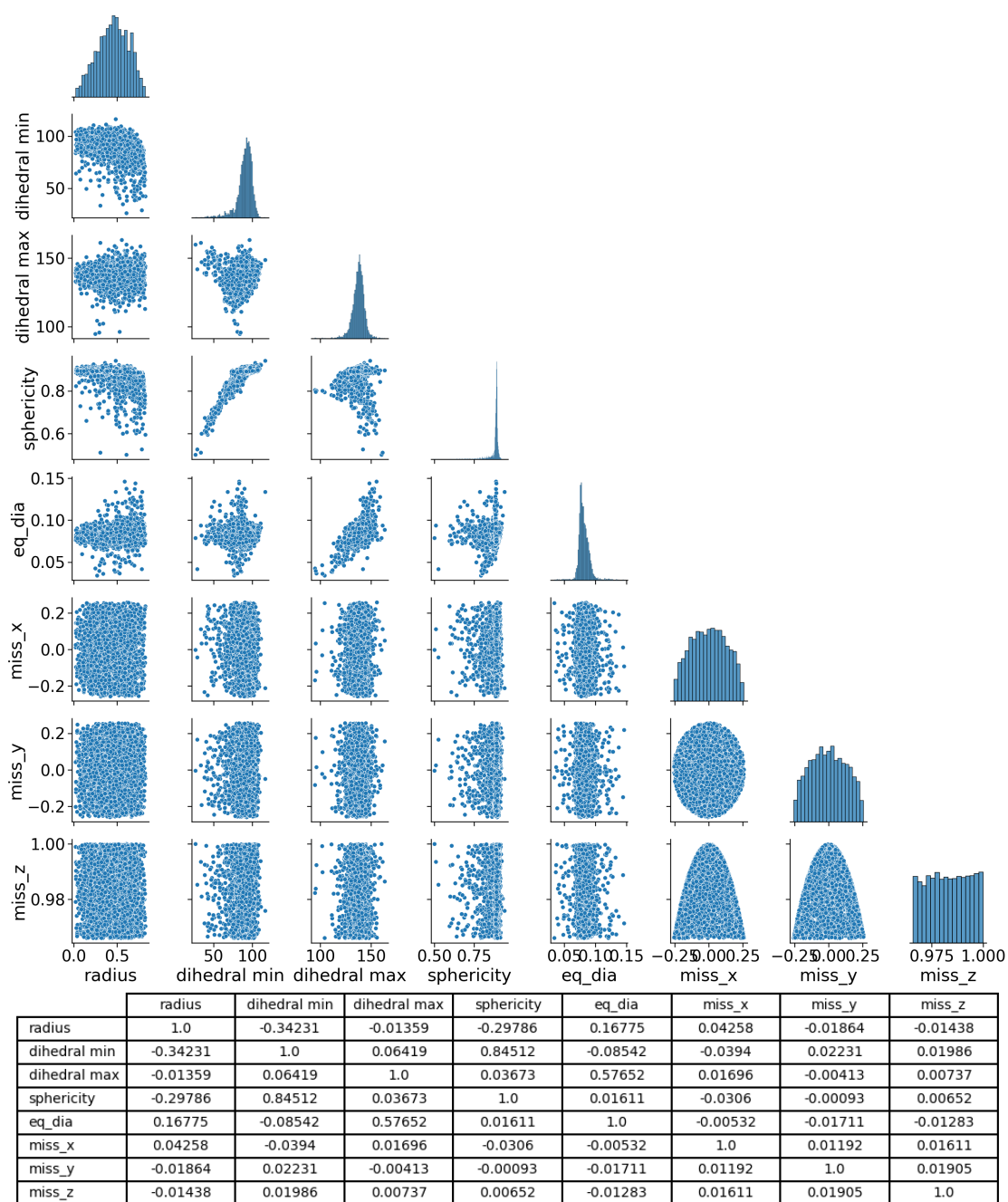


Figure 5.19: Correlation matrix for one graph in the dataset, specifically for the biggest one.

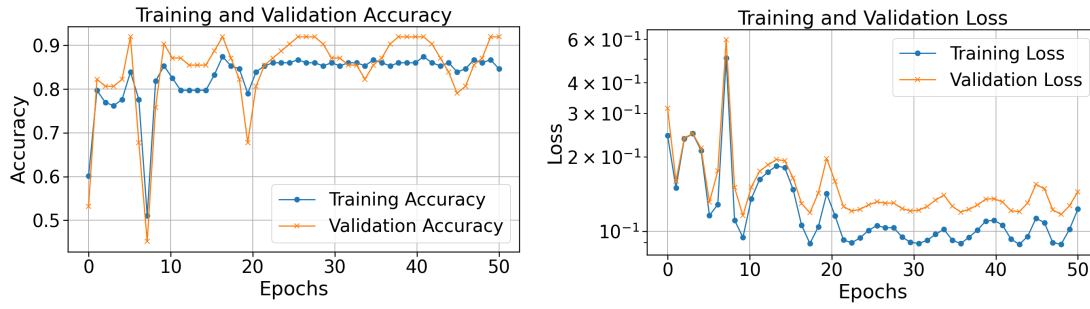


Figure 5.20: Accuracy and Loss of the 50/50 split dataset without edge updates.

True/Predicted	Class 0	Class 1
Class 0	32	2
Class 1	3	25

Table 5.13: Confusion Matrix for the 50/50 split validation set without edge updates.

minimal dihedral angle correlates strongly with sphericity which we already saw in the first dataset. Also the maximal dihedral angle correlates with the equal diameter, at least a little. Apart from these two there seems to be no real correlated features inside of the graphs. The three magnetic features stayed the same for all the grains and were therefore omitted from the graph and during training.

5.5.3 Training and optimization

It should be mentioned that previously to this dataset all the graph neural networks updated only the nodes and global features. This however meant that no information about the neighbours was transmitted to the nodes and only structural information was gained through node updates. To prove this approach two grid searches were done one with and one without edge updates. The grid search itself varied the same parameters as the grid searches done for the other datasets and initially the coercivity was split 50/50 for the range of the training set. As previously predicted this meant that there would be only one class for the coercivity in the generalisation set. For the first run where only structural information was transmitted we had the best result with a batch size of 8, neurons per layer 8, a depth of 1 and a weight decay of 10^{-5} summing to 690 parameters. The result can be seen in Figure 5.20 and from it we can see that a relatively high accuracy above 90% could be achieved for the validation set. The corresponding confusion matrix for the highest validation accuracy can be seen in Table 5.13.

Now lets look at the loss and accuracy with edge updates turned on in Figure 5.21. The parameters are nearly the same as the only change is the neurons per layer with 32 summing to 7362 parameters. What is quite interesting to see is the switch between the validation and training accuracy. Were previously the validation accuracy was higher

5 Training and optimization

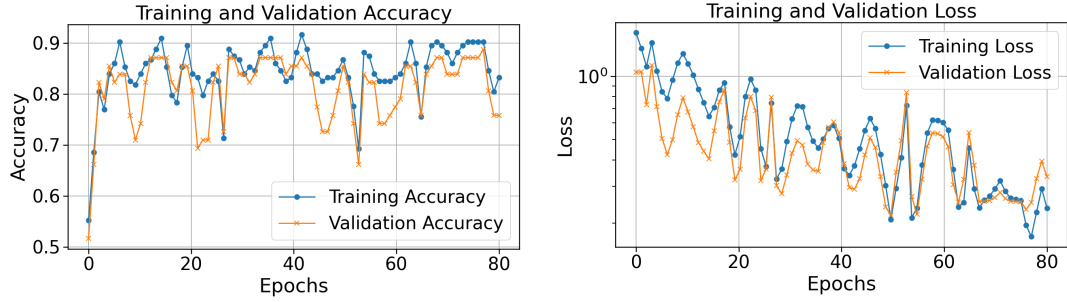


Figure 5.21: Accuracy and loss of the 50/50 split dataset with edge updates.

True/Predicted	Class 0	Class 1
Class 0	25	6
Class 1	1	30

Table 5.14: Confusion Matrix for the 50% split validation set with edge updates.

now the training accuracy reaches higher peaks. What was also interesting to see was the distance between the two decreased. For the run with 8 neurons per layer this was even more visible. What is also nice to see is that the recall is again really high with only one wrongly classified good magnet. Both shown runs achieved 100% accuracy for the generalisation dataset.

Similarly to the other datasets we looked at a 80/20 split with edge updates and could achieve an even higher accuracy for both the training and validation set. We can see the resulting accuracy and loss in Figure 5.22 and its corresponding confusion matrix in Table 5.15. The recall in this case was worse as one fourth of all good magnets got wrongly classified. The hyperparameters for this run were 32 neurons per layer, a depth of 1, a weight decay of 10^{-5} and a batch size of 16 summing to 7362 parameters.

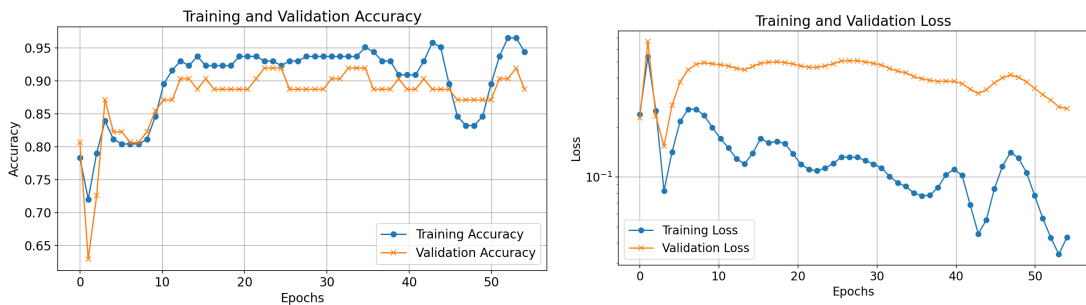


Figure 5.22: Accuracy and loss for the training and validation set with an 80/20 split coercivity.

Table 5.15: Confusion Matrix for the 80/20 split validation set

True/Predicted	Class 0	Class 1
Class 0	48	2
Class 1	3	9

Finally we split the coercivity to 20/80 to lower the threshold for good magnets, giving the generalisation set at least a few good magnets. Another grid search was started with and without edge updates to verify the generalisation capabilities of the neural network. It was found that with edge updates performed the better results with the generalisation set achieving a 75% accuracy. The corresponding accuracy and loss function can be seen in Figure 5.23 and the confusion matrix for the generalisation set in Tab 5.16. As we can see from the confusion matrix we do not reach the trivial solution of only classifying the whole set as class 0 which would yield 84% accuracy in the generalisation set. However that at least a few good magnets could be correctly classified is something that can be built upon. The parameters for this run were a batch size of 16, neurons per layer 16, a depth of 1 and 10^{-5} for weight decay summing to 2146 parameters. Alternatively we could argue that the generalisation result is actually quite good, even though we lose most of the good magnets, because we correctly classify the bigger magnets as bad magnets which was the task on hand.

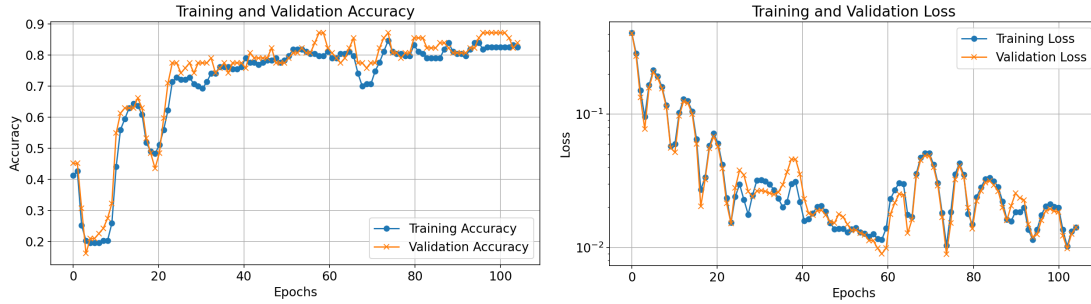


Figure 5.23: Accuracy and Loss for the 20/80 split dataset.

True/Predicted	Class 0	Class 1	True/Predicted	Class 0	Class 1
Class 0	23	5	Class 0	12	1
Class 1	3	2	Class 1	7	42

Table 5.16: Confusion Matrix for the 20/80 split generalisation set on the left and validation set on the right.

Lastly as we mentioned that early stopping was implemented we want to show the behaviour of the accuracy and loss curves when training continues. This can be seen in

5 Training and optimization

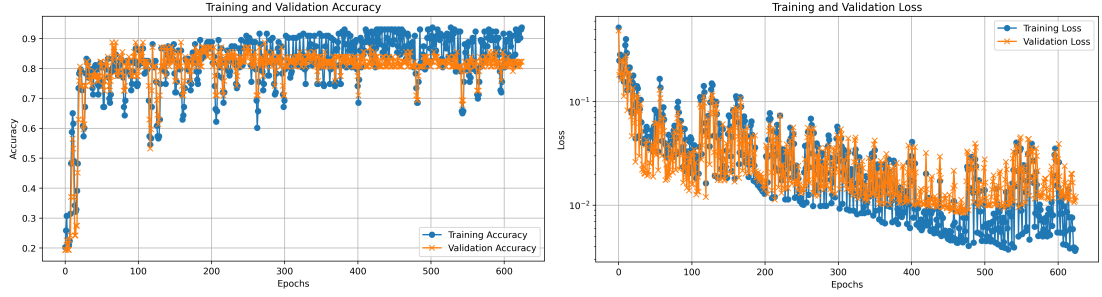


Figure 5.24: Accuracy and loss for the 20/80 split dataset however for longer training to showcase the overfitting behaviour.

Figure. [5.24](#) where we can see the standard behaviour of overfitting. The training loss as well as accuracy increase with epochs, the validation accuracy however decreases again back to the trivial solution and the validation loss stagnates.

5.6 Conclusion and outlook

What we can see from the different accuracy curves in the datasets is that variation in the data is key to a good model. With the first dataset which is only one structure over a hundred times the trained models did not achieve a good result. The graph neural network could not leverage its forte of including the structure in the training process as every structure is the same. With the second dataset we saw the difference a little variation can make achieving 100% for the validation set. However the small size of the set as well as the clear difference of coercivity depending on the average radius may have played a big role in that. Therefore it was even better to see that an accuracy above 90% for both sets could be achieved in the third dataset, with a very good recall score. In the 80/20 split coercivity case a good recall score was achieved even though the general accuracy was not as high. Finally for the fourth dataset we saw the fastest convergence in accuracy, where the other datasets needed at least 50 epochs to reach an acceptable accuracy in the 50/50 cases the fourth dataset reached a satisfying accuracy in less than 20 epochs. This we would assume is because of the high variation of structures. It was also interesting to see that just from structural information alone, as in no edge updates, a high accuracy could be achieved. As for the generalisation we saw a 100% for the case where there was only one class in the generalisation set. Even for the 20/80 split case we can see a baseline that can be built upon. Also dropout layers were mentioned but never implemented in the code as their implementation in jax and jraph is not trivial and their effect on generalisation in the case of graphs is something that should be tested, even though most of the neural networks in this thesis consist of only one hidden layer.

There are many aspects of graph neural networks that could be used, especially for the research of granular structures. In magnets for example colleagues are in the process of implementing a regression graph neural network to predict the coercivity directly, there

are also ideas of predicting the magnetisation depended on the external magnetic field to fully simulate a hysteresis loop via machine learning and ideas to find the weakest grain in a magnet so that we know which grain switches first under the influence of an external field. [EFK⁺18] Some aspects of magnets could also be included in the future as this work only focused on neodymium, iron and boron magnets. A different composition of elements for example would give us more features to work with, as three magnetic properties would finally change throughout the magnet or datasets. Also in the datasets highlighted in this thesis the grain boundary phase and therefore edge features were never fully utilised. There is one idea where a magnets boundary phase is permeated by praseodymium and the percentage of it in the specific part of the boundary phase of magnets could be used as a meaningful edge feature. As we can see there are many aspects of graph neural networks that could be worked on and their usage in research for granular structures could potentially help in faster finding new magnets so that supply chain issues regarding heavy rare earths can be circumvented. In collaboration with Toyota this work for example will be used in a dashboard. There it will classify magnets according to different criteria if they are viable to be used in electric motors or not.

Bibliography

- [Aha00] Amikam Aharoni. *Introduction to the Theory of Ferromagnetism*, volume 109. Clarendon Press, 2000.
- [Ans73] Francis J Anscombe. Graphs in statistical analysis. *The american statistician*, 27(1):17–21, 1973.
- [BBP22] Nithin Buduma, Nikhil Buduma, and Joe Papa. *Fundamentals of deep learning*. " O'Reilly Media, Inc.", 2022.
- [BCH⁺20] Silvia Bobba, Samuel Carrara, Jaco Huisman, Fabrice Mathieux, Claudiu Pavel, et al. Critical raw materials for strategic technologies and sectors in the eu. a foresight study. 2020.
- [BHB⁺18] Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.
- [BJ59] William Fuller Brown Jr. Micromagnetics, domains, and resonance. *Journal of Applied Physics*, 30(4):S62–S69, 1959.
- [BS24] K. Broadwater and N. Stillman. *Graph Neural Networks in Action*. In Action. Manning, 2024.
- [Coe10] J. M. D. Coey. *Magnetism and Magnetic Materials*. Cambridge University Press, 2010.
- [COP⁺22a] Jun Cui, John Ormerod, David Parker, Ryan Ott, Andriy Palasyuk, Scott McCall, M. Parans Paranthaman, Michael S. Kesler, Michael A. McGuire, Ikenna C. Nlebedim, Chaochao Pan, and Thomas Lograsso. Manufacturing processes for permanent magnets: Part i—sintering and casting. *JOM. Journal of the Minerals, Metals & Materials Society*, 74(4), 2 2022.
- [COP⁺22b] Jun Cui, John Ormerod, David S. Parker, Ryan Ott, Andriy Palasyuk, Scott McCall, Mariappan Parans Paranthaman, Michael S. Kesler, Michael A. McGuire, Cajetan Nlebedim, Chaochao Pan, and Thomas Lograsso. Manufacturing processes for permanent magnets: Part ii—bonding and emerging methods. *JOM. Journal of the Minerals, Metals & Materials Society*, 74(6), 2 2022.

Bibliography

- [dmh] Haiku documentation 2014; haiku documentation — dm-haiku.readthedocs.io. <https://dm-haiku.readthedocs.io/en/latest/>. [Accessed 24-02-2025].
- [EFK⁺18] Lukas Exl, Johann Fischbacher, Alexander Kovacs, Harald Oezelt, Markus Gusenbauer, Kazuya Yokota, Tetsuya Shoji, Gino Hrkac, and Thomas Schrefl. Magnetic microstructure machine learning analysis. *Journal of Physics: Materials*, 2(1):014001, 2018.
- [ES24] Lukas Exl and Sebastian Schaffer. Angewandtes maschinelles lernen, 2024.
- [ESS20] Lukas Exl, Dieter Suess, and Thomas Schrefl. *Micromagnetism*, pages 1–44. Springer International Publishing, Cham, 2020.
- [FKG⁺18] Johann Fischbacher, Alexander Kovacs, Markus Gusenbauer, Harald Oezelt, Lukas Exl, Simon Bance, and Thomas Schrefl. Micromagnetics of rare-earth efficient permanent magnets. *Journal of Physics D: Applied Physics*, 51(19):193002, 2018.
- [FN00] H. Fukunaga and H. Nakamura. Computer simulation of magnetic properties of anisotropic nanocomposite magnets. *IEEE Transactions on Magnetics*, 36(5):3285–3287, 2000.
- [GD22] Sarvesh Gharat and Yogesh Dandawate. Galaxy classification: a deep learning approach for classifying sloan digital sky survey images. *Monthly Notices of the Royal Astronomical Society*, 511(4):5120–5124, 2022.
- [Gér22] Aurélien Géron. *Hands-on machine learning with Scikit-Learn, Keras, and TensorFlow*. " O'Reilly Media, Inc.", 2022.
- [GN02] Michelle Girvan and Mark EJ Newman. Community structure in social and biological networks. *Proceedings of the national academy of sciences*, 99(12):7821–7826, 2002.
- [GOF⁺20] Markus Gusenbauer, Harald Oezelt, Johann Fischbacher, Alexander Kovacs, Panpan Zhao, Thomas George Woodcock, and Thomas Schrefl. Extracting local nucleation fields in permanent magnets using machine learning. *npj Computational Materials*, 6(1):89, 2020.
- [HCLB23] Jonathan M. Hestoffer, Marie-Agathe Charpagne, Marat I. Latypov, and Irene J. Beyerlein. Graph neural networks for efficient learning of mechanical properties of polycrystals. *Computational Materials Science*, 217:111894, 2023.
- [HG09] Haibo He and Edwardo A Garcia. Learning from imbalanced data. *IEEE Transactions on knowledge and data engineering*, 21(9):1263–1284, 2009.

- [jax] Quickstart 2014; jax documentation — docs.jax.dev. <https://docs.jax.dev/en/latest/quickstart.html>. [Accessed 24-02-2025].
- [jra] Welcome to jraph 2019s documentation! 2014; jraph 0.0.1.dev documentation — jraph.readthedocs.io. <https://jraph.readthedocs.io/en/latest/>. [Accessed 24-02-2025].
- [KEK⁺24] Alexander Kovacs, Lukas Exl, Alexander Kornell, Johann Fischbacher, Markus Hovorka, Markus Gusenbauer, Leoni Breth, Harald Oezelt, Masao Yano, Noritsugu Sakuma, Akihito Kinoshita, Tetsuya Shoji, Akira Kato, and Thomas Schrefl. Image-based prediction and optimization of hysteresis properties of nanocrystalline permanent magnets using deep learning. *Journal of Magnetism and Magnetic Materials*, 596:171937, 2024.
- [KFO⁺23] Alexander Kovacs, Johann Fischbacher, Harald Oezelt, Alexander Kornell, Qais Ali, Markus Gusenbauer, Masao Yano, Noritsugu Sakuma, Akihito Kinoshita, Tetsuya Shoji, et al. Physics-informed machine learning combining experiment and simulation for the design of neodymium-iron-boron permanent magnets with reduced critical-elements content. *Frontiers in Materials*, 9:1094055, 2023.
- [LZ20] Z. Liu and J. Zhou. *Introduction to Graph Neural Networks*. Synthesis Lectures on Artificial Intelligence and Machine Learning. Morgan & Claypool Publishers, 2020.
- [MD⁺07] Jacques E Miltat, Michael J Donahue, et al. Numerical micromagnetics: Finite difference methods. *Handbook of magnetism and advanced magnetic materials*, 2:742–764, 2007.
- [MKF⁺24] H Moustafa, A Kovacs, J Fischbacher, M Gusenbauer, Q Ali, L Breth, Y Hong, W Rigaut, T Devillers, NM Dempsey, et al. Reduced order model for hard magnetic films. *AIP Advances*, 14(2), 2024.
- [nep] Neper: Polycrystal generation and meshing 2014; neper 4.10.1 documentation — neper.info. <https://neper.info/index.html>. [Accessed 24-02-2025].
- [Öze18] Harald Özelt. *Exchange coupled ferri-/ferromagnetic composite nanomagnets*. PhD thesis, Technische Universität Wien, 2018.
- [PSBBS22] Vagelis Plevris, German Solorzano, Nikolaos P Bakas, and Mohamed El Amine Ben Seghier. Investigation of performance metrics in regression analysis and machine learning-based prediction models. In *8th European Congress on Computational Methods in Applied Sciences and Engineering (ECCOMAS Congress 2022)*. European Community on Computational Methods in Applied Sciences, 2022.
- [pyd] seaborn: statistical data visualization 2014; seaborn 0.13.2 documentation — seaborn.pydata.org. <https://seaborn.pydata.org>. [Accessed 24-02-2025].

Bibliography

- [RN10] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 4 edition, 2010.
- [sci] scikit-learn: machine learning in python 2014; scikit-learn 1.6.1 documentation — scikit-learn.org. <https://scikit-learn.org/stable/>. [Accessed 24-02-2025].
- [SF99] T. Schrefl and J. Fidler. Finite element modeling of nanocomposite magnets. *IEEE Transactions on Magnetics*, 35(5):3223–3228, 1999.
- [SGGP⁺20] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter Battaglia. Learning to simulate complex physics with graph networks. In Hal Daumé III and Aarti Singh, editors, *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 8459–8468. PMLR, 13–18 Jul 2020.
- [SLRPW21] Benjamin Sanchez-Lengeling, Emily Reif, Adam Pearce, and Alexander B. Wiltschko. A gentle introduction to graph neural networks. *Distill*, 2021. <https://distill.pub/2021/gnn-intro>.
- [SLWL⁺19] Benjamin Sanchez-Lengeling, Jennifer N Wei, Brian K Lee, Richard C Gerkin, Alán Aspuru-Guzik, and Alexander B Wiltschko. Machine learning for scent: Learning generalizable perceptual representations of small molecules. *arXiv preprint arXiv:1910.10685*, 2019.
- [SN10] Andrej-Nikolai Spiess and Natalie Neumeyer. An evaluation of r^2 as an inadequate measure for nonlinear models in pharmacological and biochemical research: a monte carlo approach. *BMC pharmacology*, 10:1–11, 2010.
- [Tes19] Sven Teske. *Achieving the Paris climate agreement goals: Global and regional 100% renewable energy scenarios with non-energy GHG pathways for + 1.5 C and + 2 C*. Springer Nature, 2019.
- [WJ22] Lisa Wang and Nikola Jovanović. Introduction to graph neural nets with jax/jraph. https://github.com/google-deepmind/educational/blob/master/colabs/summer_schools/intro_to_graph_nets_tutorial_with_jraph.ipynb, 2022. Accessed: 1.11.2024.