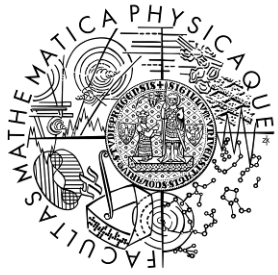


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Bc. Ondřej Dušek

Analysis of topological magnetic phases using generative machine learning models

Department of Condensed Matter Physics

Supervisor of the master thesis: RNDr. Pavel Baláž, Ph.D.

Study programme: Physics of Condensed Matter and Materials

Specialization: Physics of materials

Prague 2023

I declare that I carried out this master thesis independently, and only with the cited sources, literature and other professional sources.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 paragraph 1 of the Copyright Act.

In…..... date..... signature

I would like to thank my supervisor RNDr. Pavel Baláž, Ph.D., for his endless patience and the time he gave me. He was the one who introduced me to machine learning, and it profoundly influenced me.

I would also like to thank RNDr. Milan Straka, Ph.D., who was willing to consult the neural network architecture with me.

Title: Analysis of topological magnetic phases using generative machine learning models

Author: Bc. Ondřej Dušek

Department: Department of Condensed Matter Physics

Supervisor: RNDr. Pavel Baláž, Ph. D., FZU - Institute of Physics of the Czech Academy of Sciences

Abstract: In this thesis, we developed a model of a variational autoencoder with residual connections, trained on a dataset of skyrmion lattices. Afterwards, we explored its ability to reconstruct lattices, to encode the information describing lattices into a low-dimensional latent space, and to generate new lattices from randomly sampled points in the latent space. We have shown that the reconstruction squared error between the lattice used as an input and the reconstructed lattice correlates with the number of defects in the lattice. This could be used for detecting defects in lattices. We have demonstrated that the model is able to encode physical properties such as the topological charge Q or mean magnetization M_Z of these lattices into the latent space. This comparison was done for multiple variational autoencoders, differing in the weight used to multiply their Kullback-Leibler divergence loss during the training.

Keywords: machine learning, neural networks, magnetic skyrmions

Contents

Introduction

During the last decade, machine learning, particularly generative models, has experienced significant advancements – not only because of the accessibility of more powerful hardware, but also due to the development of more sophisticated techniques. This thesis focuses on the application of these innovative techniques to the study of magnetic configurations, specifically within the realm of the two-dimensional Heisenberg model on a square lattice. We will develop a generative model called the variational autoencoder, and use it to examine skyrmion lattices.

The primary objective of this thesis is threefold; Firstly, we will develop the architecture of this machine learning model and test whether it is able to understand the intricacies of skyrmion lattices. Secondly, we will train this model and search for the best set of parameters that maximizes the performance of the model. And thirdly, we will evaluate how the model performs and explore practical applications of the trained models.

The goal is to have a model, that can extract useful information from the lattices and encode it in a much smaller space, which can help find anomalies in the lattices, and that can possibly even generate new, theoretically plausible skyrmion lattices.

1 Theoretical part

In this work, we will examine properties of ferromagnetic materials. We can imagine such a system as a 2-dimensional lattice which has a unit spin (described using θ and φ angles) in each of its nodes. This system can be described with a classical Heisenberg Model, which will be discussed in subsequent chapters. Besides the Heisenberg Hamiltonian, we will also include the term describing the magnetic field influence and the Dzyaloshinskii–Moriya interaction (also known as Asymmetric exchange), which is caused by a spin-orbit interaction due to an absence of inversion symmetry. In a system described by a Hamiltonian containing these three terms, there can exist multiple different phases, notably the skyrmion phase, the spiral phase and the ferromagnetic phase, discussed further.

1.1 Magnetic skyrmions

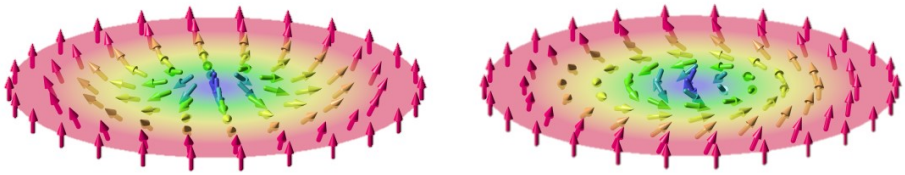
At the beginning, let's describe what a magnetic skyrmion is. The surface of some magnetic material has magnetization, which can be represented using unit vectors. Skyrmions can exist in this material; they are a vortex-like configuration of those spins [1]. A skyrmion is a localized and stable topological soliton, and it was first predicted to exist by the British physicist Tony Skyrme during the 1960's in the context of particle physics [2]. That a skyrmion is a topological object means that there is no continuous transformation that could project the skyrmion on a uniform configuration [1]. It means that its spin configuration is protected when interacting with other skyrmions.

A skyrmion is tied to the notion of a topological charge (also called topological quantum number or skyrmion number). The topological charge of a skyrmion always has a non-zero integer value. It is defined as

$$N_{Sk} = \frac{1}{4\pi} \int \boldsymbol{m}(\boldsymbol{r}) \cdot \left(\frac{\partial \boldsymbol{m}(r)}{\partial x} \times \frac{\partial \boldsymbol{m}(r)}{\partial y} \right) dx\, dy, \tag{1.1}$$

where N_{Sk} is the topological charge and $\boldsymbol{m}(\boldsymbol{r})$ is the spin value in the given point. The topological charge describes how many times the skyrmion configuration wraps around a unit sphere.

A visualization of two of the most common types of skyrmions is shown in Picture 1.1. Both Néel type skyrmions and Bloch type skyrmions have $Q = -1$.



Picture 1.1 – Néel type skyrmion (left) and Bloch type skyrmion (right). Taken from [3] under CC BY-SA 3.0.

Using radial coordinates $\boldsymbol{r} = r \cdot (cos\phi, sin\phi)$, we can transform the equation (1.1) to

$$N_{Sk} = \frac{1}{4\pi}\int_0^{\infty}\int_0^{2\pi}\frac{\partial\Phi(\phi)}{\partial\phi}\frac{\partial\theta(r)}{\partial r}\,d\phi\,dr, \tag{1.2}$$

which can be further simplified to

$$N_{Sk} = m \cdot p = \left[\frac{1}{2\pi}\Phi(\phi)\right]_{\phi=0}^{2\pi} \cdot \left[-\frac{1}{2}cos\theta(r)\right]_{r=0}^{\infty}, \tag{1.3}$$

where the first term is called the vorticity $m \in \mathbb{Z}$ and the second term is called the polarity $p = \pm 1$ [2]. For each unit spin vector, we then have

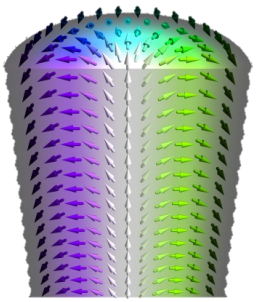
$$\boldsymbol{n}(\boldsymbol{r}) = (sin\theta(\boldsymbol{r})cos\Phi(\boldsymbol{r}), sin\theta(\boldsymbol{r})sin\Phi(\boldsymbol{r}), \cos\theta(\boldsymbol{r}))\,. \tag{1.4}$$

One additional parameter characterizing skyrmions is the helicity γ – we can calculate it from the azimuthal angle Φ and the vorticity m as

$$\gamma = m \cdot \varphi - \Phi. \tag{1.5}$$

The helicity can be understood as a phase factor, and is different between Néel type and Bloch type skyrmions. For the Néel type skyrmion, helicity is $\gamma = 0$ and for the Bloch type skyrmion it is $\gamma = \pm\frac{\pi}{2}$ [4].

If we had a 3-dimensional lattice, skyrmions would typically extend in a tube-like manner, as indicated in Picture 1.2.



Picture 1.2 – Extension of a skyrmion in 3 dimensions. Taken from [2].

Besides skyrmions, different non-trivial particle-like structures do exist, such as antiskyrmions ($N = 1, m = -1$), merons and antimerons ($N = \pm\frac{1}{2}$), biskyrmions and others [4].

1.2 Stabilizing mechanisms for magnetic skyrmions

The notion of a spin lattice has been already introduced at the beginning of the theoretical part. A skyrmion lattice can be described using the Heisenberg model; the most important energy contribution to such a system is the symmetric Heisenberg exchange interaction. The Hamiltonian describing this interaction is

$$H_{exchange} = -\frac{1}{2}\sum_{ij} J_{i,j}\boldsymbol{S}_i \cdot \boldsymbol{S}_j, \tag{1.6}$$

where $\boldsymbol{S}_i = \frac{\boldsymbol{\mu}_i}{|\boldsymbol{\mu}_i|}$ is a unit vector in the direction of the corresponding magnetic moment $\boldsymbol{\mu}_i$. $J_{i,j}$ denotes the exchange integral, which is related to the charge distribution between atoms [4]. The equation can be simplified to

$$H_{exchange} = -\frac{1}{2}J\sum_{<ij>}\boldsymbol{S}_i \cdot \boldsymbol{S}_j, \tag{1.7}$$

where we include only the interaction between nearest neighbors. The exchange integral J is a constant in that case. J is responsible for how the neighboring spins are arranged, for $J > 0$ we have a ferromagnetic (parallel) ordering and for $J < 0$ we have an antiferromagnetic (antiparallel) ordering.

However, the exchange interaction is not enough to explain the existence of a skyrmion phase on a magnetic lattice. The second interaction that needs to be included is the Dzyaloshinskii–Moriya interaction (often referred to as DMI). The energy contribution of the DMI interaction is

$$H_{DMI} = \frac{1}{2}\sum_{ij}(\boldsymbol{S}_i \times \boldsymbol{S}_j) \cdot \boldsymbol{D}_{ij}. \tag{1.8}$$

Spins $\boldsymbol{S}_i$ and $\boldsymbol{S}_j$ are the same as those in the exchange interaction equation and $\boldsymbol{D}_{ij}$ is a Dzyaloshinskii–Moriya vector, that characterizes the interaction.

Once again, if we include only the interaction between nearest neighbors, the equation can be simplified into

$$\boldsymbol{D} \cdot \sum_{<ij>}[\boldsymbol{S}_i \times \boldsymbol{S}_j], \tag{1.9}$$

where we sum only between nearest neighbors. Since the Dzyaloshinskii–Moriya vector has a C_v symmetry, for the vector $\boldsymbol{u}_{ij}$ between two magnetic moments i and j

(from i to j) it holds that if $\boldsymbol{u}_{ij} = (\pm 1, 0, 0)$, then $\boldsymbol{D} = (0, \pm D, 0)$, and if $\boldsymbol{u}_{ij} = (0, \pm 1, 0)$, then $\boldsymbol{D} = (\pm D, 0, 0)$ where D is a single scalar parameter.

The Dzyaloshinskii–Moriya interaction is a spin-orbit interaction that exists due to the lack of inversion symmetry.

Lastly, we will also consider the influence of a magnetic field on the system (Zeeman term). We will assume the field is perpendicular to the lattice, thus $\boldsymbol{B} = (0, 0, B)$. Under such an assumption, its energy is

$$B \sum_i S_i^Z, \tag{1.10}$$

where we once again sum over all the lattice spins.

Besides the Dzyaloshinskii–Moriya interaction, other stabilizing mechanisms are the frustrated exchange interaction, the four-spin exchange interaction and magnetic dipole coupling [2][5][6][7].

1.3 Lattice defects

We can describe lattice defects as deviations from the strictly periodic arrangement of atoms in a solid. There exist extrinsic defects, which are caused by impurities, and intrinsic defects, which are caused by a disarrangement of atoms in the solid [8]. We are interested in intrinsic defects. We can also group various types of defects according to their dimensionality. Thus, we have point defects, line defects (dislocations), planar defects, and bulk defects.

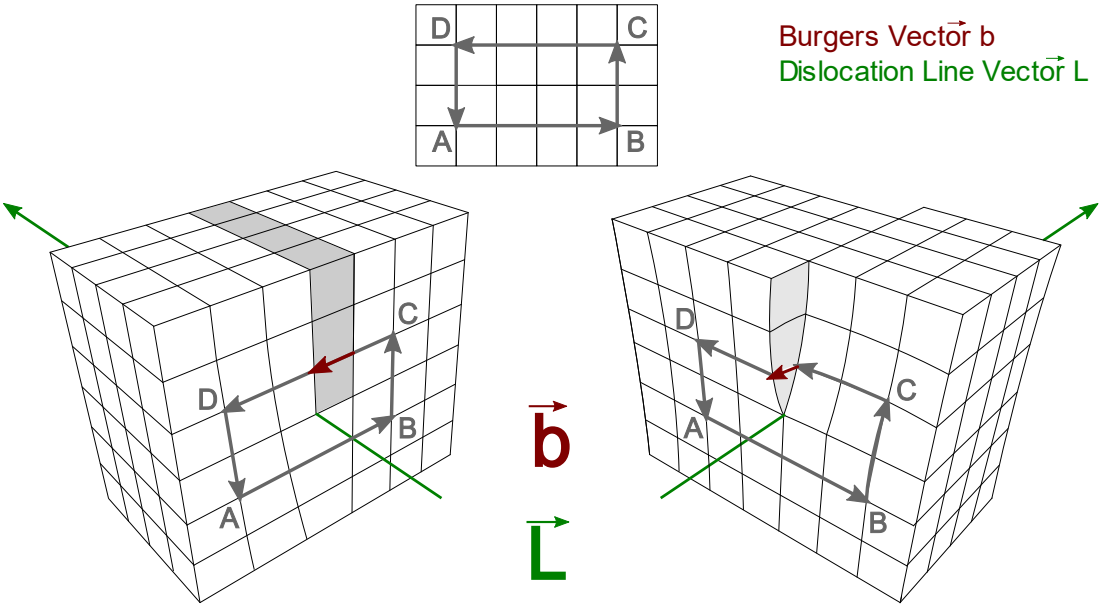
Point defects are for example vacancies and interstitials. A vacancy is a missing atom from a lattice site, and an interstitial is an atom that occupies space between lattice sites. A vacancy can be of two types – the first one is the Frenkel defect, which consists of a vacancy and a nearby interstitial atom from the vacancy, and the other is the Schottky defect, which is just the vacancy – the original atom is missing (for example by diffusion to the surface) [8][9]. The equilibrium number n of vacancies, under the assumption that $n \ll N$, where N is the number of atoms, is

$$\frac{n}{N} \cong \exp\left(-\frac{E_v}{k_B T}\right), \tag{1.11}$$

where E_v is the energy to take an atom from a lattice site inside the crystal to the surface, k_B is the Boltzmann constant and T is the temperature [9]. Thus, higher temperature leads to an increased number of defects.

Line defect (dislocation) occurs when a whole line of lattice points is perturbed. Line defects in crystals are a consequence of shear stress. There exist edge dislocations and

screw dislocations. An edge dislocation could be understood as inserting an extra plane of atoms into a part of the lattice [9]. A screw dislocation differs from an edge dislocation in the sense that the boundary between slipped and unslipped parts of the crystal is parallel to the slip direction, instead of being perpendicular to it. Both dislocations are illustrated in Picture 1.3.



Picture 1.3 – An illustration of an edge dislocation (left) and a screw dislocation (right). Burgers vectors $\boldsymbol{b}$ characterizing both dislocations are also shown. Taken from [8].

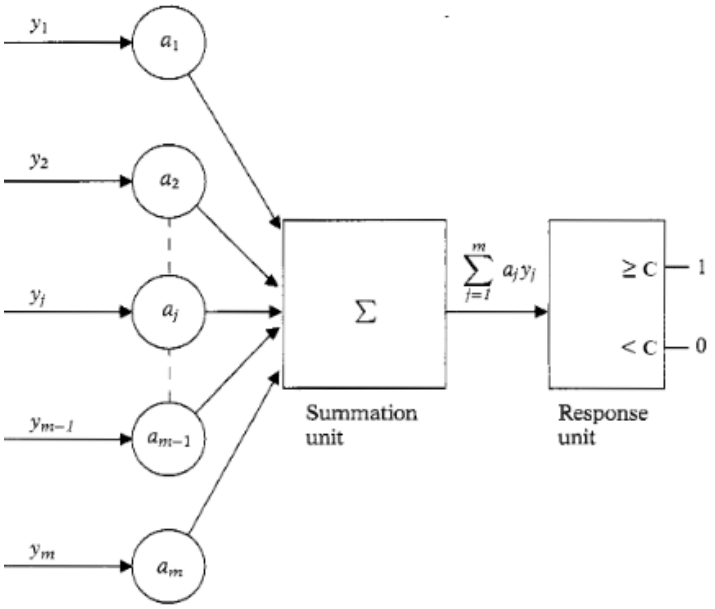
Dislocations are characterized using a Burger's vector $\boldsymbol{b}$, which gives the magnitude and direction of the slipping process. We can first draw a closed curve around the original site of the dislocation, but without the dislocation itself. Let's say that this curve has the form of a rectangle, given by points A, B, C, D and grey lines in Picture 1.3. When we introduce the dislocation, the curve will be deformed along with the lattice, and the vector needed to close the curve is exactly Burger's vector $\boldsymbol{b}$ [9].
Examples of planar defects are grain boundaries and twin boundaries. Bulk defects are precipitates, vacancy clusters, pores and cracks.

2 Used methods

In this thesis, we used machine learning to study skyrmion lattice configurations. Namely, we utilized variational autoencoders, which are a special type of unsupervised generative neural networks. In this chapter, I will introduce the basics of neural networks and their applications, then, convolutional neural networks, which are utilized in variational autoencoders, will be discussed, and the chapter will conclude with the description of variational autoencoders.

2.1 Basics of Neural Networks

The basic principle of how neural networks function can be demonstrated on a simple concept called a perceptron [10] (an idealization of a neuron), which is shown in Picture 2.1.



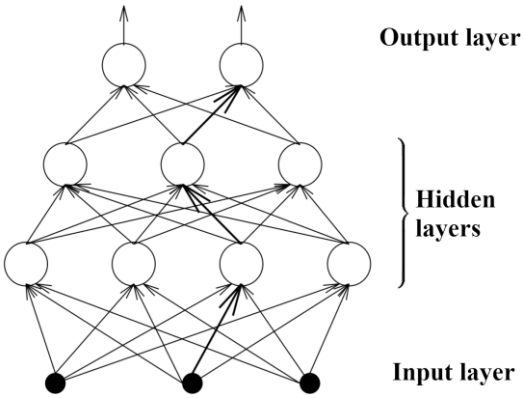
Picture 2.1 – a scheme describing a perceptron. Taken from [10].

Arrows $\{y_1, y_m\}$ denote the inputs into the perceptron (such as values of spins at x, y, z lattice positions). Each of those inputs is then multiplied by the corresponding weight $\{a_1, a_m\}$ and in the summation unit, those values and the constant C from the response unit are added together, resulting in the sum

$$A = \sum_{i=1}^{m} a_i y_i + C. \tag{2.1}$$

Finally, the perceptron returns the value 1 if $A \geq 0$ and returns 0 if $A < 0$. The constant C in (2.1) is often being referred to as a bias.

We would like to construct a network out of these perceptrons in such a way, that multiple perceptrons take as inputs values describing the object we are interested in, and the outputs of these perceptrons are used as inputs for multiple subsequent perceptrons – we repeat that enough times, until we use outputs of the last batch of perceptrons to give us the information we need. Practically, it is useful to think of perceptrons as being in layers, as it is indicated in Picture 2.2. As described, we can see the input layer (inputs into the layer are parameters of the examined object), the hidden layers, and the output layer (returns the information we seek).



Picture 2.2 – a scheme describing a simple neural network. Taken and modified from [11].

We want the neural network to be trainable – at the beginning, we create an architecture with weights that are initialized randomly, but after we feed into it a dataset containing examples of objects we would like to model, the algorithm needs to be able to tweak the values of the weights in such a way, that the neural network performs the task we intended, whether it is a classification, regression, generation, or something else.

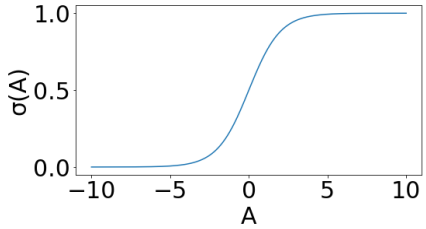
However, the concept of the perceptron shown in Picture 2.1 has a problem that makes it unusable for that - it needs to be modified. Neural networks are being trained using a method called gradient descent [12], which is an algorithm for optimizing the neural network in the space of a loss function $E(\boldsymbol{\theta})$, where $\boldsymbol{\theta}$ are weights of the model and the loss function is a scalar function that evaluates the performance of the model. Training will be discussed in detail in further chapters, but we can already make a simple observation. Since gradient descent is a gradient based method that utilizes $\nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta})$, in which we change the weights iteratively by incremental steps, we require that a small change of a random weight results only in a small change of the output of

the network. The perceptron, like we described it above, does not fulfill this premise, since the Heaviside step function $H = \begin{cases} 1, x \geq 0 \\ 0, x < 0 \end{cases}$ that is applied to the sum (2.1) does not have a continuous first derivative in 0. Thus, a different function (called an activation function) has to be used.

One of the first activation functions used was the logistic sigmoid function

$$\sigma(A) = \frac{1}{1+\exp(-A)}, \tag{2.2}$$

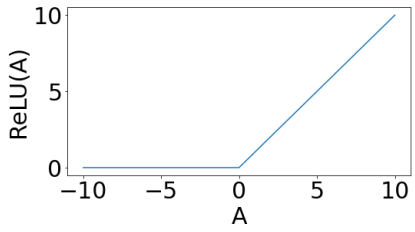
where A is the sum (2.1).



Picture 2.3 – sigmoid activation function

Since $\lim_{A\to\infty} \sigma(A) = 1$ and $\lim_{A\to-\infty} \sigma(A) = 0$, it in both limits converges to the Heaviside step function, however, it has a continuous derivative in 0. Later, different activation functions started being used, most notably $\tanh(A) = 2\sigma(2A) - 1$ (the result of tweaking the sigmoid to be symmetrical and making the derivation in zero 1) and especially ReLU (rectified linear unit):

$$ReLU(A) = \max(0, A) \tag{2.3}$$



Picture 2.4 – ReLU activation function

Once we use a reasonable activation function instead of the Heaviside step function, we can talk of a neuron instead of a perceptron, and it is these neurons that form a neural network.

Besides empirical evidence, how can we be sure that such a structure is able to model what we want? According to the Universal Approximation Theorem [13], any bounded, non decreasing function $\varphi(x)$: $\mathbb{R} \to \mathbb{R}$ can be used as an activation function

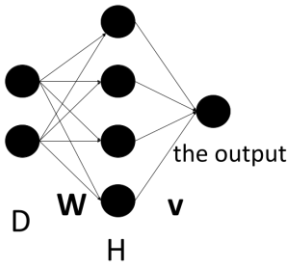
after a single layer of neurons of size H with D inputs, in such a way, that we are able to approximate a continuous function f to an arbitrary precision (given H is large enough). For any $\varepsilon > 0$ and any continuous function $f: [0,1]^D \to \mathbb{R}$, there exists $H \in \mathbb{N}$, $\boldsymbol{v} \in \mathbb{R}^H$, $\boldsymbol{b} \in \mathbb{R}^H$ and $\boldsymbol{W} \in \mathbb{R}^{D \times H}$ such, that if we denote

$$F(\boldsymbol{x}) = \boldsymbol{v}^T \varphi(\boldsymbol{x}^T \boldsymbol{W} + \boldsymbol{b}) = \sum_{i=1}^{H} v_i \varphi(\boldsymbol{x}^T \boldsymbol{W}_{*,i} + b_i), \tag{2.4}$$

where φ is applied element-wise, then for all $\boldsymbol{x} \in [0,1]^D$:

$$|F(\boldsymbol{x}) - f(\boldsymbol{x})| < \varepsilon. \tag{2.5}$$

A sketch of the network is shown in Picture 2.5. $\boldsymbol{W}$ is the matrix of weights that linearly calculates values of H neurons in the hidden layer from D input values, **b** is the vectors of constants added as C in equation (2.1), and $\boldsymbol{v}$ is a vector of transformation between the hidden layer after the activation function φ was applied and the output node.



Picture 2.5 – a picture describing the simple neural network used in the Universal Approximation Theorem

Proof is outside of the scope of this work; however, it utilizes the fact that if a function is continuous on a closed interval, it can be approximated by a sequence of lines to an arbitrary precision. Proofs exist even for ReLU, which is an unbounded function [14]. Nowadays, predominantly ReLU and its modifications (ReLU multiplied by the cumulative distribution of the standard normal distribution – GeLU, ReLU which is slightly increasing for $A = (-\infty,\ 0)$ – Leaky ReLU, etc.) are being used [15].

2.2 Neural network training

In the previous chapter, we have introduced the concept of a neural network. Now, let's describe in detail the training process.

When solving usual machine learning problems, the neural network learns from a large dataset of observations describing the task at hand. We have a train dataset, which is

used to train the neural network, and a test dataset, which is not used during training, but on which we evaluate how the model performs after it was trained.

Machine learning is also often divided into two categories, supervised machine learning and unsupervised machine learning. In supervised machine learning, observations in both datasets are also accompanied by labels which describe the corresponding observation. Neural networks are then usually trained to assign the correct label to a previously unseen observation. In unsupervised machine learning, observations in both datasets are not accompanied by labels. For example, large language models or certain generative models use unsupervised ML. Since we are building a generative model, the task of this thesis falls into this category.

Before the training of a neural network begins, its weights are randomly initialized. Numerous weight initializations methods are being used; the most common one and the one we are using is the Glorot uniform initialization [16] – a matrix of weights $\boldsymbol{W} \in \mathbb{R}^{D\times H}$ is initialized as $\boldsymbol{W} \sim \boldsymbol{U}\left[-\sqrt{\frac{6}{D+H}}, \sqrt{\frac{6}{D+H}}\right]$, where $\boldsymbol{U}$ denotes a uniform distribution from the given interval. Bias vectors are initialized as zeros.

The model is trained by minimizing the loss function, such as the mean squared error, between samples from the training data and network outputs.

Mean squared error loss function for N samples $\left(\boldsymbol{x}^{(1)}, y^{(1)}\right), \ldots, \left(\boldsymbol{x}^{(N)}, y^{(N)}\right)$ from the train dataset is

$$L_{MSE} = \frac{1}{N}\sum_{i=1}^{N}\left(f\left(\boldsymbol{x}^{i}; \boldsymbol{\theta}\right) - y^{(i)}\right)^2, \tag{2.6}$$

where $\boldsymbol{x}^i$ and y^i denotes an observation and its label from the dataset, $f\left(\boldsymbol{x}^{i}; \boldsymbol{\theta}\right)$ is the output of the model and $\boldsymbol{\theta}$ are the parameters (weights and biases) of the model. Unsupervised machine learning differs from a supervised machine learning in the sense, that we train the model on unlabelled data; the model learns without any guidance or instructions. It is often used for anomaly detection or clustering.

The loss function is calculated during a forward propagation regime, during which values of neurons in the first layer are calculated, those are used to calculate values of neurons in the second layer, etc., until we reach the output layer that calculates $f\left(\boldsymbol{x}^{i}; \boldsymbol{\theta}\right)$. As we could see in equation (2.4), the hidden layer $\boldsymbol{h} \in \mathbb{R}^M$ can be calculated from the preceding layer (or an input) $\boldsymbol{x} \in \mathbb{R}^N$ as

$$\boldsymbol{h} = f(\boldsymbol{x}^T \boldsymbol{W}^{(h)} + \boldsymbol{b}^h), \tag{2.7}$$

where $\boldsymbol{W} \in \mathbb{R}^{N\times M}$ is a matrix of weights and $\boldsymbol{b} \in \mathbb{R}^M$ is a vector of biases.

Once the loss function is calculated, we minimize it by changing weights and biases using gradient descent

$$\boldsymbol{\theta} \leftarrow \boldsymbol{\theta} - \alpha \nabla_{\theta} E(\boldsymbol{\theta}), \tag{2.8}$$

where $E(\boldsymbol{\theta})$ is

$$E(\boldsymbol{\theta}) = \mathbb{E}_{(x,y)\sim\hat{p}_{data}} L(f(\boldsymbol{x};\boldsymbol{\theta}), y) \tag{2.9}$$

and α is a constant called a learning rate.

The most currently used version of the gradient descent algorithm is the minibatch stochastic gradient descent – we take only *N* random samples (a batch) from the dataset and apply the algorithm (2.8), and we repeat it until the network saw all samples from the dataset. Once the whole dataset was used, the first epoch of training was completed and we begin cycling through the dataset using minibatch SGD anew. The other two variants are when our batch is the whole dataset or only a single sample.

The gradient $\nabla_{\theta} E(\boldsymbol{\theta})$ in (2.8) is calculated using an algorithm called backpropagation. For the loss L and a weight/bias j in i-th layer w_{ij}, $y_j = \varphi\left(\sum_{k=1}^{N} w_{kj} x_k\right) = \varphi(net_j)$ is the neuron's output and φ its activation, using the chain rule of derivatives we get [11]

$$\frac{\partial L}{\partial w_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial w_{ij}} = \frac{\partial L}{\partial y_j}\frac{\partial y_j}{\partial net_j}\frac{\partial net_j}{\partial w_{ij}}. \tag{2.10}$$

Those partial derivations are calculated from the last layer to the first, since we need results for the farther layers to calculate partial derivations for the front ones – thus the name backpropagation algorithm. One of the reasons ReLU activation is used is that $\frac{\partial y_j}{\partial net_j}$ can be calculated faster [15], even though we have to define the derivative in zero to be 0.

Besides the mean squared error loss function, the cross-entropy loss function is sometimes used – models using it perform better on some tasks. It is given by

$$L_{cross-E} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M}\left[y_j ln\hat{y}_j + (1-y_j) ln(1-\hat{y}_j)\right], \tag{2.11}$$

where *N* is the number of samples in the batch, *j* indexes output neurons, y_j the target label and $\hat{y}$ is the model's output [17]. It comes from the information theory and expresses the amount of surprise when a random variable is sampled.

2.3 Convolutional neural networks

In the previous sections, how neural networks function was explained only on networks with fully connected layers, such as is the one shown in Picture 2.2. When working with data that have a spatial or temporal dependency, such as image data,

speech or our magnetic lattices, it is often advantageous to use a different type of layers – convolutional layers.

Imagine the input $\boldsymbol{I}$ into the neural network is a tensor with 3 dimensions – x and y coordinates and the c channel dimension, which, in the case of the magnetic lattice, can contain for example θ and φ values characterizing the unit vector in each site. A convolutional layer uses the discrete convolution operation, in which the input is convolved with a tensor $\boldsymbol{K}$ called a kernel/filter:

$$(\boldsymbol{K} \star \boldsymbol{I})_{x,y,o} = \sum_{m,n,c} \boldsymbol{I}_{x+m,y+n,c} \boldsymbol{K}_{m,n,c,o}, \tag{2.12}$$

where the dimension o sets the number of channels of the output tensor [18]. Elements of the kernel are the trainable parameters.

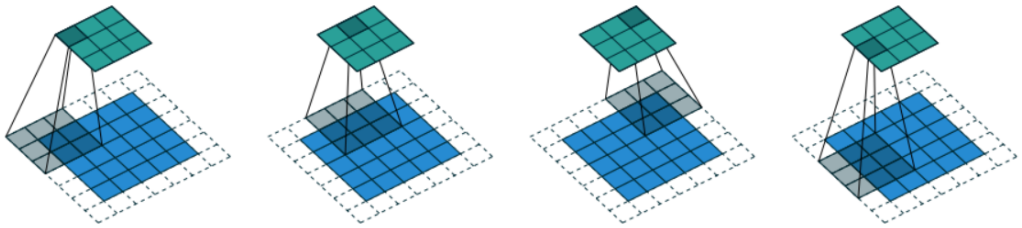
Convolutional layers have two main advantages over fully connected layers. Firstly, they allow us to drastically reduce the number of trainable parameters in the layer, which speeds up the training process significantly and allows us to build deeper neural networks. Secondly, once the neural network learns to discern a certain pattern on a specific part of the input picture, it is able to discern it everywhere – it is translation invariant [12].

The most common filter size is $(3, 3)$, and the number of channels of the output tensor o is being generally increased the deeper in the neural network the convolutional layer is. Because the convolution operation as written in the equation (2.12) decreases the x and y dimensions (convolving $\boldsymbol{I}$ with dimensions $(200, 200, 2)$ with $\boldsymbol{K}$ with dimensions $(3, 3, 2, 2)$ results in a tensor with dimensions $(198, 198, 2)$. Thus, a padding is often added, meaning that before the convolution is computed, the x and y dimensions are extended by one on both sides by a vector filled with zero values.

Besides convolutional layers that keep the dimensionality of the input, sometimes we intentionally need to reduce the dimensionality. It is possible to do that with convolutional layers that utilize strides S – we modify the equation (2.12):

$$(\boldsymbol{K} \star \boldsymbol{I})_{x,y,o} = \sum_{m,n,c} \boldsymbol{I}_{x \cdot S+m,y \cdot S+n,c} \boldsymbol{K}_{m,n,c,o}. \tag{2.13}$$

Stride $S \in \mathbb{N}$ denotes that an output pixel is calculated only for every S-th pixel. For reducing dimensionality, usually $S = 2$ is used, which reduces both x and y dimensions by half. A convolution with strides is depicted in Picture 2.6.

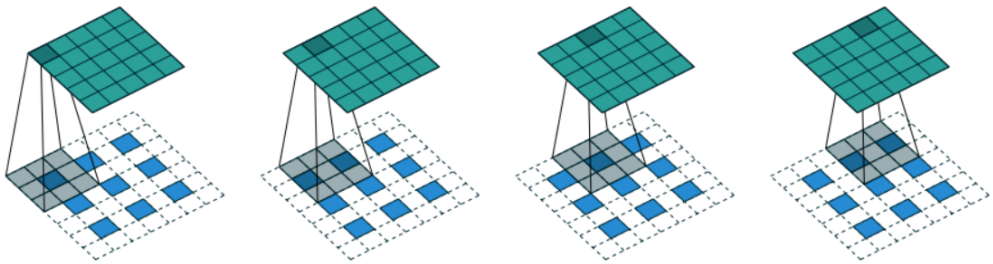


Picture 2.6 – convolution of a $(3,3)$ kernel over a $(5,5)$ input with $(1,1)$ padding and the stride $S = 2$. The input are the blue elements, the output are the green elements, the filter is grey and the padded values are white. Taken from [19].

Another type of convolutional layers are layers that utilize transposed convolution. Those layers are used when we need to increase the dimensionality, instead of decreasing it. We can obtain such a layer by running the equation (2.13) in a backward pass, which leads to the equation [18]

$$\frac{\partial \boldsymbol{L}}{\partial \boldsymbol{I}_{x,y,c}} = \sum_{x' \cdot S + m = x} \sum_{y' \cdot S + n = y} \sum_{o} \frac{\partial \boldsymbol{L}}{\partial (\boldsymbol{K} \star \boldsymbol{I})_{x',y',o}} \boldsymbol{K}_{m,n,c,o}. \tag{2.14}$$

A more intuitive notion of transposed/upscaling convolution offers Picture 2.7.



Picture 2.7 – The transpose of convolving a $(3,3)$ kernel over a $(5,5)$ input with $(1,1)$ padding and stride 2. Taken from [19].

2.4 Regularization mechanisms

It makes sense that if a very small neural network is used on a very difficult task, the network might not have enough capacity to solve the task – we say the model is underfitting (not only do we have a bad performance on a test dataset, but the model is not doing well even on a train dataset). It might not be intuitive that having an unnecessarily large model for an easy task leads to a similar problem, called overfitting – such a model may opt to memorize specific portions of each training sample rather

than understanding their distinct features. This approach, while effective on the train dataset, results in poor performance when the model encounters new, unseen data in the test dataset. A model that is generalizing well, on the other hand, learns to identify useful features instead of relying on memorization. Good generalization usually means that the error on the train dataset is similar as on the test dataset.

Numerous techniques that aim to reduce the generalization error (while not necessarily reducing the training error, sometimes the opposite is true) were developed. Here, only the ones used in this thesis will be mentioned.

The most powerful regularization technique is data augmentation. When working with convolutional networks that process image-like data, very often creating new samples for the train dataset by taking original samples and applying operations such as blurring, tilting, shifting, zooming them usually enhances the model performance [20]. Since we work with data that were generated by a Monte Carlo simulation, we cannot use most of those operations. However, because our lattices have periodic boundary conditions, shifting the lattice by an integer multiple of the distance between two lattice points in both *x* and *y* directions and rotating the lattice by an integer multiple of $\frac{\pi}{2}\ rad$ is possible. That way, we can greatly enlarge the size of the dataset and make it harder for the model to memorize individual samples.

Another regularization technique is lowering the model capacity by making it smaller. One could also put a bottleneck into the neural network – there could be two powerful parts of a neural network connected by a layer that has intentionally a small number of neurons, which forces the network to extract only the key information from the input. This is actually a core mechanism of autoencoders and variational autoencoders, which will be discussed in the following chapters.

Another mechanism that enhances the model performance is batch normalization [21]. Batch normalization is usually applied on a layer of neurons right before an activation function. During training, it learns the variance and mean of the layer outputs and normalizes them as

$$\hat{x}_i = BN(x_i) = \frac{x_i - \mathbb{E}[x_i]}{\sqrt{Var[x_i]}}, \tag{2.15}$$

where x_i are the values of the neurons and $\hat{x}_i$ are their normalized values [12]. Batch normalization reduces the need for bias in the equation (2.7), thus it can be rewritten as

$$\boldsymbol{h} = f\big(BN(\boldsymbol{W}\boldsymbol{x})\big). \tag{2.16}$$

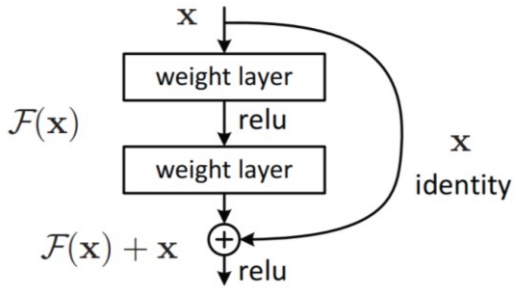
It improves the learning process when training deeper neural networks and help when using activation functions with saturating non-linearities [21].

2.5 Residual neural networks

A residual neural network is a type of a neural network, that contains residual connections (also called skip connections). Residual connections were an answer to the degradation problem of convolutional neural networks – with an increasing depth, after a certain threshold, not only was the test accuracy decreasing, but the accuracy on the train dataset was decreasing as well, suggesting that this decrease of accuracy is not caused by overfitting [22]. The decrease of training accuracy was surprising, because one can imagine that if additional layers that only do identity mapping were added to a shallow convolutional model, its accuracy would not decrease. It is speculated that if an identity mapping in a part of a neural network was truly optimal, it is difficult for the network to fit it using a stack of nonlinear layers [22]. Residual blocks replace (a set of) convolutional layers, and instead of the usual mapping $\mathcal{H}(\boldsymbol{x})$, they are trying to fit the mapping

$$\mathcal{H}(\boldsymbol{x}) = \mathcal{F}(\boldsymbol{x}) + \boldsymbol{x}. \tag{2.17}$$

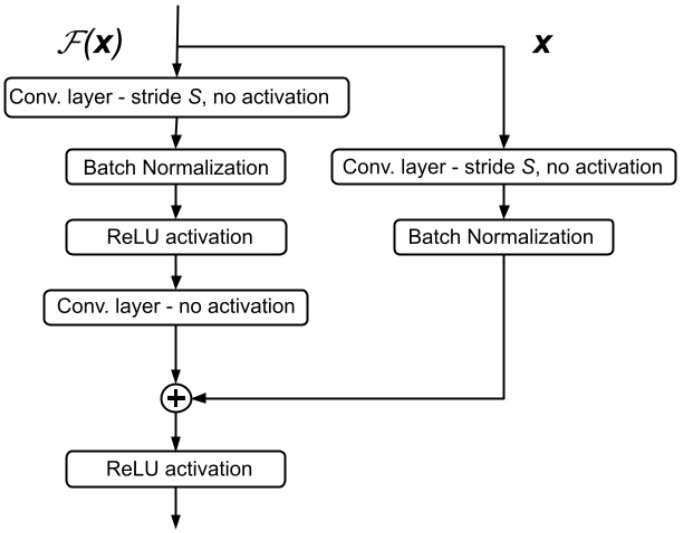
$\boldsymbol{x}$ in the equation (2.17) is the residual connection, $\mathcal{F}(\boldsymbol{x})$ represents the hidden layers. If its output was zero, $\mathcal{H}(\boldsymbol{x})$ would by default be an identity mapping. A more intuitive depiction is provided in Picture 2.8.



Picture 2.8 – A set of layers depicting a residual block, fitting the mapping (2.17). Taken from [22].

Residual connections significantly increase the accuracy of very deep convolutional networks (with tens of layers or more). Multiple types of residual blocks that differ in the form of the hidden layers are being used nowadays [23]. It is possible to create upsampling and downsampling residual blocks [24]. An example of such a

downsampling block is depicted in Picture 2.9. In that case, the residual connection $\boldsymbol{x}$ contains a hidden layer, but without any activation function, so it is only a linear transformation.

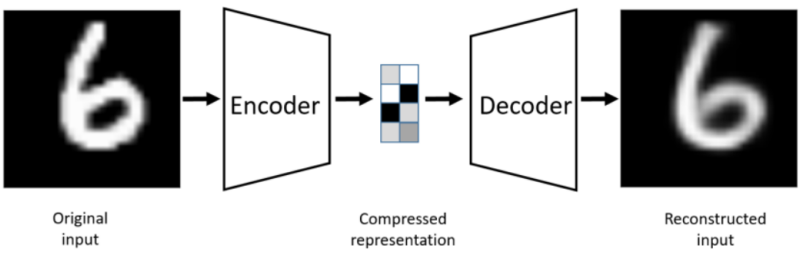


Picture 2.9 – A downsampling residual block with a stride $S \geq 2$

An upsampling residual block can be done analogously – only instead of the convolutional layer with the stride S and no activation in both $\boldsymbol{x}$ and $\mathcal{F}(\boldsymbol{x})$, we would have a transposed convolution with the same properties.

2.6 Autoencoders and variational autoencoders

Autoencoders belong to the category of unsupervised machine learning and are a useful tool for extracting an optimal representation of the training data [25]. The model consists of two parts – an encoder, which is a neural network that reduces the dimensionality of the input and tries to compress it into a small number of useful features, and a decoder, which takes the representation given to it by the encoder and tries to reconstruct the original input. A depiction of such a model is shown in Picture 2.10.



Picture 2.10 – A representation of an autoencoder. Taken from [26].

The space of the outputs of the encoder – the compressed representations - is called the latent space and is usually denoted as $\boldsymbol{z}$. It needs to have a lower dimensionality than is the dimensionality of the input. The task autoencoders perform is trying to understand the distribution $P(\boldsymbol{x})$, describing observations $\boldsymbol{x}$. The distribution can be rewritten as

$$P(\boldsymbol{x}) = \sum_{\boldsymbol{z}} P(\boldsymbol{z})P(\boldsymbol{x}|\boldsymbol{z}). \tag{2.18}$$

We use the neural network to estimate the conditional probability $P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})$. Ideally, the encoder would be just a reversed decoder - $P_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$, however, that is not technically possible, so we approximate $P_{\boldsymbol{\theta}}(\boldsymbol{z}|\boldsymbol{x})$ by a different model, an encoder, which is a trainable $Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$.

The loss function used to train the model is usually the cross-entropy loss or the mean squared error loss [27]. It is called the reconstruction loss and it measures the error between the original input and the reconstructed input. If we augment the input $\boldsymbol{x}$ as $\boldsymbol{x} + \boldsymbol{\varepsilon}$, where $\boldsymbol{\varepsilon}$ is a small noise, the network can be trained to perform denoising [28]. Even though such an autoencoder can function as a generative model in the sense, that if we send it an input, it generates a similar one, it has troubles generating completely new representations. We could take a random vector from the latent space and generate an output from it, but there is no guarantee that for that exact vector from the latent space, there is going to be a meaningful representation. Variational autoencoders address these limitations.

2.7 Variational autoencoders

The variational autoencoder differs from a standard autoencoder by the way it encodes inputs into the latent space. An encoder in a standard autoencoder generates $\boldsymbol{z}$ – an encoder in a variational autoencoder generates a distribution of $\boldsymbol{z}$'s. It is one of the mechanisms that ensures that the whole latent space generates meaningful representations.

How does that work? We represent each dimension of $\boldsymbol{z}$ as a normal distribution

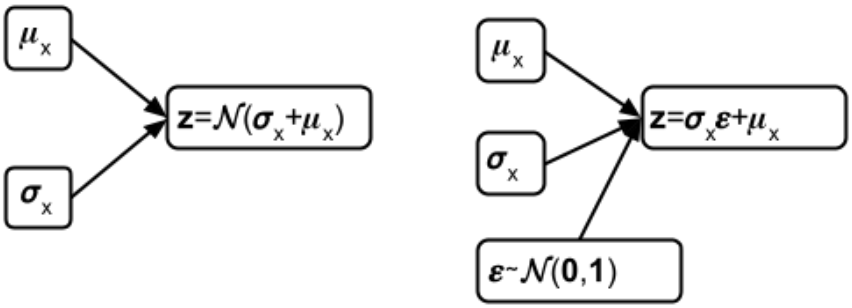
$$\boldsymbol{z} = \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\sigma}^2), \tag{2.19}$$

described by its mean value $\boldsymbol{\mu}$ and its variance $\boldsymbol{\sigma}^2$. Both $\boldsymbol{\mu}$ and $\boldsymbol{\sigma}^2$ are represented by a layer in the encoder output. During training, a random $\boldsymbol{z}$ from the normal distribution is sampled. Because the backpropagation algorithm would not be able to propagate

gradients back through this sampling layer, a modification is needed. This modification is called the reparameterization trick [26] and it is portrayed in Picture 2.11. We sample a vector $\boldsymbol{\varepsilon}$ from the normal distribution $\mathcal{N}(\mathbf{0}, \mathbf{1})$ with zero mean and unit variance. This vector is then used to obtain $\boldsymbol{z}$ for a given $\boldsymbol{x}$ as

$$\boldsymbol{z} = \boldsymbol{\sigma_x} \odot \boldsymbol{\varepsilon} + \boldsymbol{\mu_x}. \tag{2.20}$$

Since we understand $\boldsymbol{\varepsilon}$ as just another input into the sampling layer, we are able to backpropagate gradients through it.



Picture 2.11 – A sampling layer without the reparameterization trick (on the left) and a sampling layer with the reparameterization trick (on the right)

One has to be careful when creating the layer containing variances $\boldsymbol{\mu}$. They cannot be negative, so one either needs to use an exp activation function, or the model can be trained to learn $\log(\boldsymbol{\mu})$ and then the exponential can be applied to it when calculating $\boldsymbol{z}$ afterwards.

The loss function used for training variational autoencoders differs from that of a standard autoencoder. Training the decoder alone without employing the encoder using the cross-entropy loss as indicated in the equation (2.21)

$$\log P_{\boldsymbol{\theta}}(\boldsymbol{x}) = \log\ \mathbb{E}_{P(\boldsymbol{z})}[P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] \tag{2.21}$$

is not possible. This is because sampling a random $\boldsymbol{z}$ would lead to a huge variance, hindering the training process [29]. Thus, utilizing the encoder and Jensen's inequality for distributions, we can modify the equation (2.21) as

$$\log P_{\boldsymbol{\theta}}(\boldsymbol{x}) \geq \mathbb{E}_{Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}\left[\log P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z}) + \log \frac{P(\boldsymbol{z})}{Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})}\right], \tag{2.22}$$

which leads to the loss function [26]

$$L(\boldsymbol{\theta}, \boldsymbol{\varphi}; \boldsymbol{x}) = \mathbb{E}_{Q_{\varphi}(\boldsymbol{z}|\boldsymbol{x})}[-\log P_{\boldsymbol{\theta}}(\boldsymbol{x}|\boldsymbol{z})] + D_{KL}\big(Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x}) \| P_{\boldsymbol{z}}\big), \tag{2.23}$$

where the first term is our reconstruction loss and the second term is the Kullback–Leibler divergence, also known the latent loss, which optimizes the latent space by

trying to make the encoder latent space distribution $Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$ as close as possible to the normal distribution $P_{\boldsymbol{z}} = \mathcal{N}(\mathbf{0}, \mathbf{1})$ [29]. $Q_{\boldsymbol{\varphi}}(\boldsymbol{z}|\boldsymbol{x})$ is parametrized as $\mathcal{N}(\boldsymbol{z}|\boldsymbol{\mu}, \boldsymbol{\sigma}^2; \mathbf{1})$. The latent loss is usually weighted; too high a latent loss forces the normal distribution describing some random sample $\boldsymbol{x}$ spread out over the whole latent space. We want it to cover only the nearest area around its $\boldsymbol{z}$ coordinates. Conversely, too low a latent loss might lead to some regions in the latent space not outputting samples corresponding to the given domain.

2.8 Principal Component Analysis

Principal component analysis, often referred to as PCA, is a statistical technique, often used for dimensionality reduction, feature extraction or data visualisation. Its main goal is to transform a high-dimensional dataset into a lower-dimensional representation while losing as little information as possible [30]. We will use it to examine the latent space of the variational autoencoder.

It is an orthogonal linear transformation $\boldsymbol{Y} = \boldsymbol{XW}$, where $\boldsymbol{X}$ represents the data, $\boldsymbol{W}$ is a matrix of weights, and $\boldsymbol{Y}$ is the resulting orthogonal transformed matrix of principal components. Those principal components are calculated sequentially, from the first to the last. The first principal component is

$$PC_1 = \underset{\|\omega\|=1}{\operatorname{argmax}}\{Var(\boldsymbol{X}\omega)\}, \tag{2.24}$$

where ω is the weight vector and we are maximizing the variance of the projected data. Subsequent principal components are calculated as

$$PC_k = \underset{\|\omega\|=1, \omega \perp PC_1, \dots, PC_{k-1}}{\operatorname{argmax}} \{Var(\boldsymbol{X}\omega)\}. \tag{2.25}$$

The resulting principal components are linearly uncorrelated. We can see that the ordering means that the first component explains most of the variance of the data and the last component explains the least. Hence the use for dimensionality reduction – often, it is possible to use only the N most important principal components and discard the rest.

3 Own work

3.1 Training and testing data

In this thesis, we worked with square spin lattices of the size $N \times N = 200 \times 200$ that had already been prepared in advance. They were created using Monte Carlo

simulations in Uppsala Atomistic Spin Dynamics package (UppASD) [31], utilizing the Heisenberg Hamiltonian

$$H = -J\sum_{<ij>}\boldsymbol{S}_i \cdot \boldsymbol{S}_j - \boldsymbol{D'} \cdot \sum_{<ij>}\left[\boldsymbol{S}_i \times \boldsymbol{S}_j\right] - B'\sum_i S_i^Z. \quad (3.1)$$

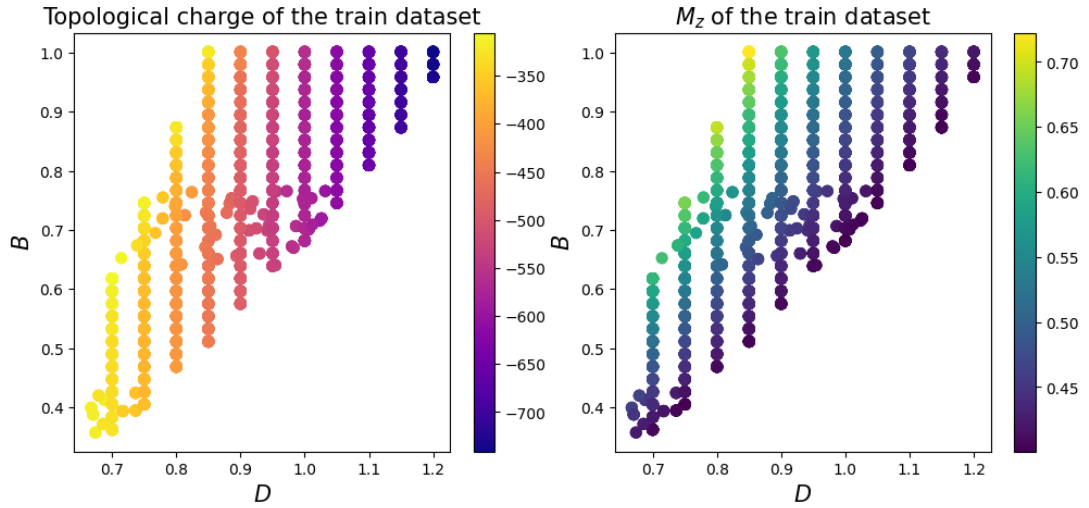
Thus, the exchange interaction, Dzyaloshinskii–Moriya interaction and the influence of the perpendicular magnetic field were included. We only consider the nearest neighbors interaction. We assumed that the $\boldsymbol{D'}$ and B' distribution on the lattice was uniform. Lattices were simulated for different values of the parameters $\boldsymbol{D'}$ and B'. The parameter J was removed by dividing the Hamiltonian (3.1) by it, resulting in the dimensionless equation

$$\widetilde{H} = -\sum_{<ij>}\boldsymbol{S}_i \cdot \boldsymbol{S}_j - \boldsymbol{D} \cdot \sum_{<ij>}\left[\boldsymbol{S}_i \times \boldsymbol{S}_j\right] - B\sum_i S_i^Z, \quad (3.2)$$

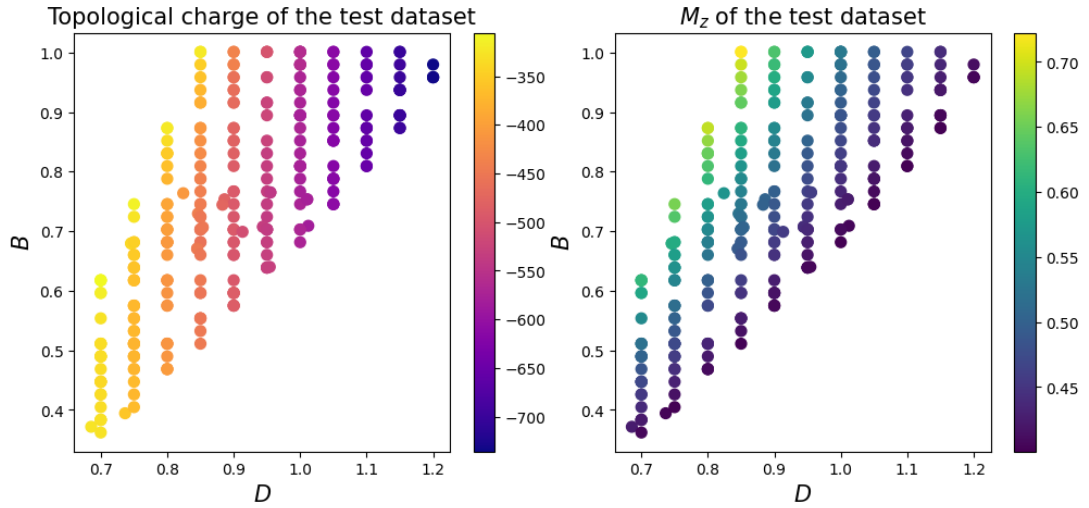
where $\boldsymbol{D} = \frac{\boldsymbol{D'}[Joule]}{J}$ and analogously $B = \mu_B \frac{B'[T]}{J}$. μ_B is the Bohr magneton and $B'[T]$ is the magnetic induction.

We set $J = 1$. Since the Dzyaloshinskii–Moriya vector has C_v symmetry (shown in the equation (1.9)), D is a single scalar parameter. The temperature was expressed as $T = k_B \frac{T'[K]}{J}$, where k_B is the Boltzmann constant and $T'[K]$ is the temperature. The lattices were simulated for $T = 0.006$. We will work with these parameters.

The dataset of lattices used for training and testing the neural network models contains 2860 lattices. 2574 of them were used for training those models and 286 were used for testing (unseen during training). Both the train and test datasets contain only skyrmion lattices; no spiral or ferromagnetic phases were present. They are depicted in Picture 3.1 and Picture 3.2, along with the average topological charge Q and the mean magnetization in the z-axis M_Z of lattices corresponding to each set of (D, B). To each couple of parameters (D, B) in those pictures corresponds five independent configurations.



Picture 3.1 – A scatter plot depicting D and B, along with the topological charge and M_z of phase diagram points, from which were sampled the train dataset lattices



Picture 3.2 – A scatter plot depicting D and B, along with the topological charge and M_z of phase diagram points, from which were sampled the test dataset lattices

3.2 Motivation

Our goal is to create a variational autoencoder that should be able to generate new magnetic lattices. The motivation follows; simulating magnetic lattices using Monte Carlo simulations is a computationally expensive process, which limits the quantity of lattices we are able to obtain. Having a model that is able to generate those lattices much faster is therefore an attractive prospect.

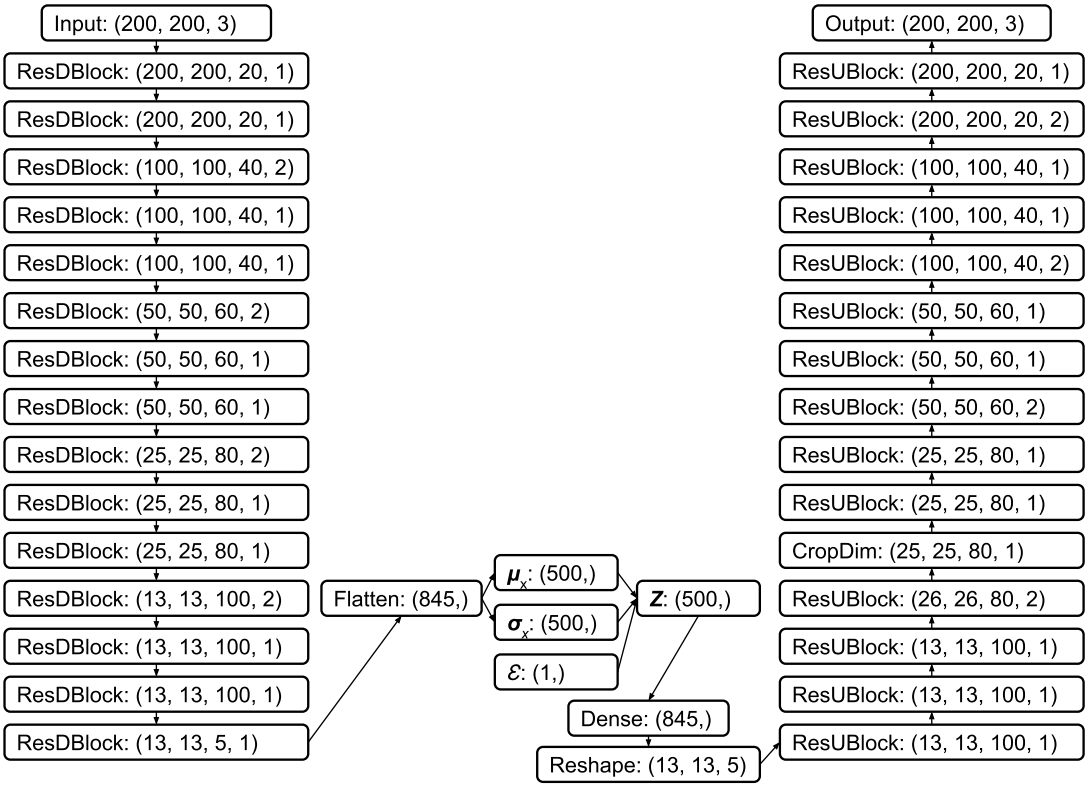
Generating new lattices utilizes the decoder part of the variational autoencoder, which generates lattices from their reduced representation in its latent layer. However, the encoder part of the model is also useful. We can use it to compress an existing lattice into a space of a much lower dimensionality, while retaining almost all of the original information. Then, instead of analysing properties of the original lattice, we can work with this reduced representation.

Another possible use-case is the combination of those two possibilities. We can use the encoder part of the model to generate a reduced representation in the latent space from an existing lattice, sample a different vector from the latent space, which is however in a close proximity to the reduced representation, and generate a new lattice from it using the decoder. The resulting lattice should have properties similar to the original lattice, but should differ slightly.

Once we create the variational autoencoder, we would like to examine how many of those tasks it is able to perform. This means that we need to understand, how well it is able to reconstruct the input lattices and whether the model introduces any artifacts into the generated lattices during reconstruction. We will also need to examine the properties of the latent space and finally, the model's ability to create lattices from any given point in the latent space. These are the questions we aim to answer.

3.3 Architecture and training of the variational autoencoder

When we first approached this task, we tried to start with a simple VAE architecture, consisting of a few convolutional layers along with simple Max-Polling and upsampling layers, similarly to the model in [32]. However, it quickly became apparent that this approach would not work – the model would not train. It could be perhaps because we were working with significantly larger lattices or because those lattices contained many more skyrmions, which made the task more complex. Thus, we decided to use significantly more sophisticated models, utilising tens of blocks with residual connections, allowing us to make a very deep neural network. Initially, we also trained the model on lattices, where the unit spins were expressed using θ and φ angles, but x, y, z coordinates seemed to perform better (despite making the lattice representation even bigger). Eventually, we converged to an architecture was able to learn and which is shown in Picture 3.3. Let's name it VAE 1.



Picture 3.3 – A scheme describing the neural network VAE 1. *ResDBlock* is a shortcut for the residual downsampling block, *ResUBlock* is a shortcut for the residual upsampling block, $\boldsymbol{\mathcal{Z}}$ is the VAE sampling layer. The network starts with the *Input* and ends with the *Output* layer. The first three numbers in all brackets denote the dimensions of the output of the corresponding layer, the 4th number in the bracket is the size of the stride *S*.

We have implemented the neural network models in the programming language Python and besides standard Python libraries such as NumPy or matplotlib, we used Tensorflow and Keras [12].

Since we train the network on 200×200 lattices of spins, where each spin has x, y, z coordinates, the network's input is in fact a tensor of the dimensionality $\mathbb{R}^{200,200,3}$. The residual downsampling block has the same architecture as the one shown in Picture 2.9, the residual upsampling block is analogous to the downsampling block, but the first convolutional layer in both $\boldsymbol{x}$ and $\mathcal{F}(\boldsymbol{x})$ uses a transposed convolution. The kernel used in convolutional layers in those blocks and all other convolutional layers as well has the size $(3, 3)$. The layer named *Flatten* only performs an operation that reshapes a tensor $\mathbb{R}^{m,n,o}$ to $\mathbb{R}^{m \cdot n \cdot o}$. The *Dense* layer denotes a standard fully-connected

layer. The *CropDim* layer denotes an operation that transforms a tensor $\mathbb{R}^{m,n,o}$ to $\mathbb{R}^{j,k,l}$ where $m \geq j$, $n \geq k$, $o \geq l$ by dropping values in the corresponding dimensions. $\boldsymbol{\mu}_x$, $\boldsymbol{\sigma}_x$ and ε are all as shown in Picture 2.11. The *Output* layer uses the $tanh$ activation function instead of the standard $ReLU$; $\tanh(x) \in (-1,1)$, which is also the range of possible values for x, y, z coordinates of unit spins. A part of the $Output$ layer is also unit normalization; x, y, z are rescaled so that the vector (x, y, z) has a unit length.

The loss function was $L = L_{MSE}(\boldsymbol{x}_{in}, \boldsymbol{x}_{out}) + C_{KL} \cdot L_{KL}(\boldsymbol{z})$, where the first term is the mean squared error reconstruction loss between the input and the output lattice and the second term is the Kullback–Leibler divergence loss (2.23) multiplied by the C_{KL} constant, which we will call the KL weight parameter. The optimizer used was the ADAM algorithm [33]. The model was trained with a batch size of 32 for 1000 epochs and the dimension of the latent space, as seen in Picture 3.3, was 500.

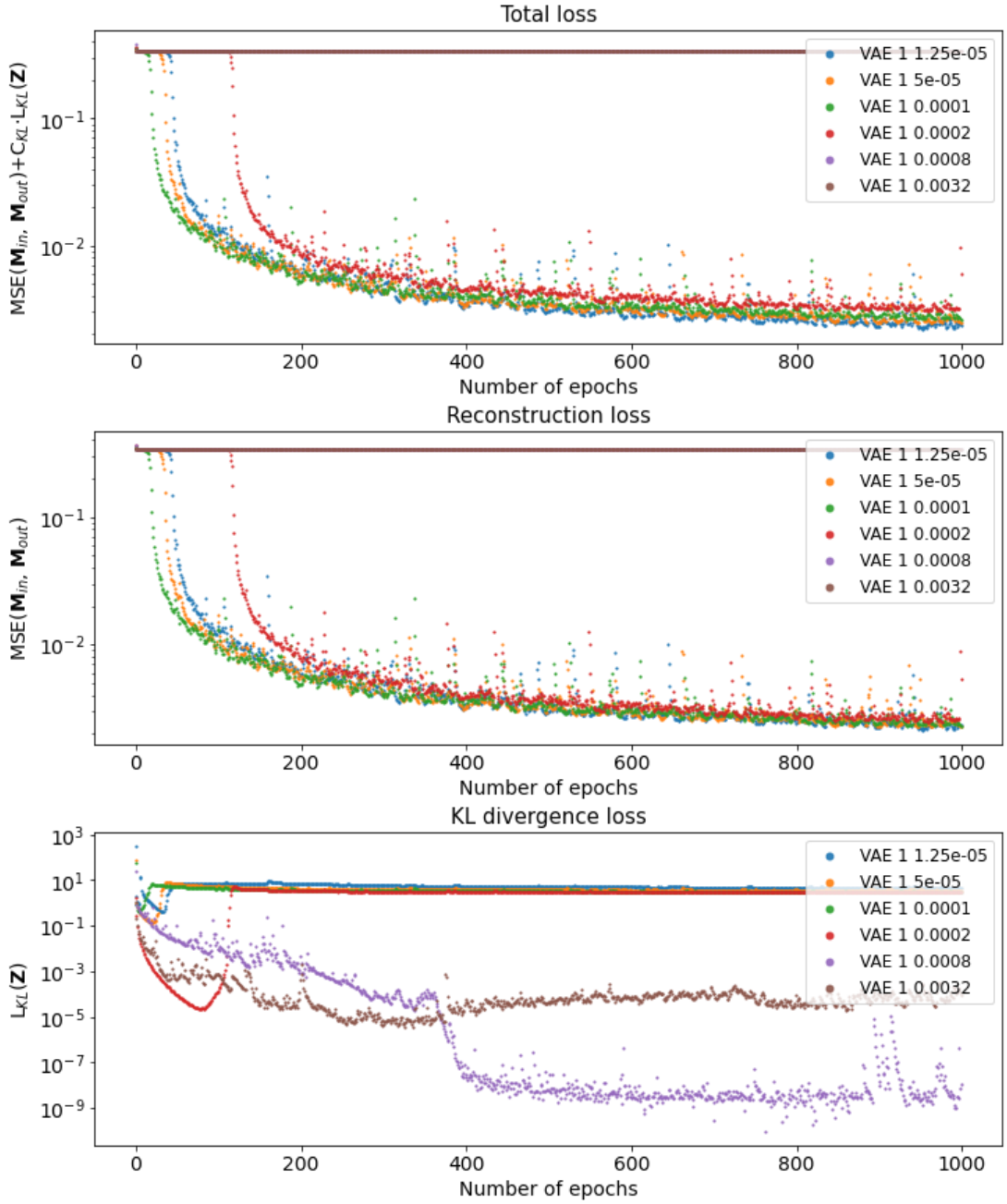
We trained six such models with six different KL weight parameters $C_{KL} \in \{1.25 \cdot 10^{-5}, 5 \cdot 10^{-5}, 1 \cdot 10^{-4}, 2 \cdot 10^{-4}, 8 \cdot 10^{-4}, 32 \cdot 10^{-4}\}$. The names of those models and the appropriate KL weight parameters are shown in Table 3.1. Each model is denoted according to its KL weight parameter, which is the only property that differs between them.

Model name	**KL weight C_{KL}**
VAE 1 1.25e-05	$1.25 \cdot 10^{-5}$
VAE 1 5e-05	$5 \cdot 10^{-5}$
VAE 1 0.0001	$1 \cdot 10^{-4}$
VAE 1 0.0002	$2 \cdot 10^{-4}$
VAE 1 0.0008	$8 \cdot 10^{-4}$
VAE 1 0.0032	$32 \cdot 10^{-4}$

Table 3.1 – VAE 1 model names and the corresponding KL weight parameters

Training progress of all VAE 1 models is shown in Picture 3.4. The model was trained on the train dataset of 2574 lattices. We can see that models VAE 1 0.0008 and VAE 1 0.0032 completely failed to train – their reconstruction error did not decrease almost at all. This was definitely caused by the KL weight parameter being too large, as can be seen when observing the KL divergence loss; compared to other models, VAE 1 0.0008 and VAE 1 0.0032 have the KL divergence loss orders of magnitude smaller. Also, the KL weight parameter is the only thing that differs between the models. Other

models did manage to generalize, VAE 1 0.0002 took around 180 epochs until the reconstruction loss began significantly decreasing, and the rest of those models took higher tens of epochs. VAE 1 0.0002 took longer than the rest possibly also due to the higher KL weight parameter.



Picture 3.4 – Mean train dataset losses each epoch for all VAE 1 models during training

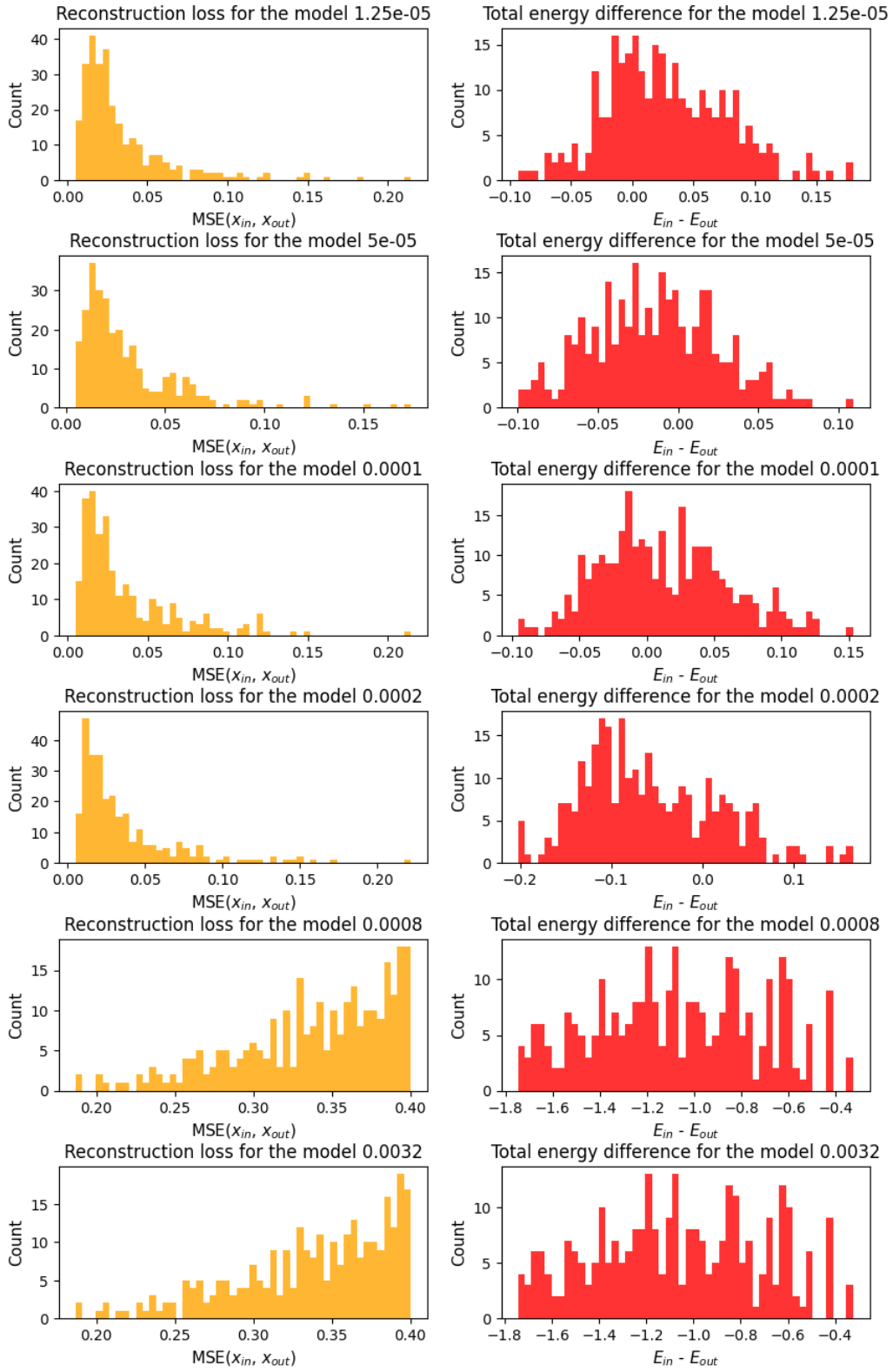
3.4 Evaluation of performance of the variational autoencoder

Now that the models are trained, let's have a general overview of how they perform. For that, we will need the test dataset. While there are many things that we can test on a variational autoencoder, the most important is its ability to accurately reconstruct previously unseen data.

We can test this by taking lattices $\boldsymbol{x}_{in}$ from the test dataset, feeding them into the encoder, getting their latent representations in $\boldsymbol{\mathcal{Z}}$, feeding those into the decoder, getting the reconstructed lattices $\boldsymbol{x}_{out}$ and measuring the mean squared error between these input and output lattices. There is a small twist however; in the equation (2.20), which describes how we obtain $\boldsymbol{\mathcal{Z}}$ during training, were sampled elements of the vector $\boldsymbol{\varepsilon}$ from a normal distribution. This time, to obtain as accurate representations as possible, we need to set to whole vector $\boldsymbol{\varepsilon}$ to zero. This will be the case during the whole testing. We will have a use for $\boldsymbol{\sigma}_{\boldsymbol{x}}$ only once we will try generating new lattices from randomly sampled points in the latent space.

Besides looking at the mean squared errors between the test dataset lattices and the reconstructed lattices, we can also look at the difference between their energies. The reconstruction loss depicts how well the models reconstructs the lattice and the energy difference indicates, whether other properties of the reconstructed lattice are similar to the input one.

Histograms that capture both the mean squared error between test dataset lattices and their reconstructions, and the differences between their energies, are shown in Picture 3.5.



Picture 3.5 – Histograms depicting the mean squared error and the energy difference between test dataset lattices and their reconstructions from VAE 1 models 1.25e-05, 5e-05, 0.0001, 0.0002, 0.0008 and 0.0032.

Histograms for VAE 1 0.0008 and VAE 1 0.0032 models in Picture 3.5 confirm what we have observed about them when examining training curves in Picture 3.4. There, we could see that their reconstruction loss on the train dataset did not converge. Here, we can see that neither did their reconstruction loss on the test dataset – since the test dataset reconstruction loss is $L_{MSE} = \frac{1}{N}\sum_{i=0}^{N} MSE(\boldsymbol{x}_{i,in}, \boldsymbol{x}_{i,out})$, meaning the average value from the histograms on Picture 3.5. Their reconstruction loss on the test dataset is more than an order of magnitude larger than the reconstruction loss of other VAE 1 models.

Reconstruction losses on the test dataset of all VAE 1 models are shown in Table 3.2. We can see that VAE 1 5e-05 has the lowest reconstruction loss. VAE 1 0.0008 and VAE 1 0.0032 models will not be included in further analyses, and when referring to "all VAE 1 models", those two models will not be encompassed in the reference. Thanks to them, we see, that $C_{KL} \geq 0.0008$ leads to the VAE 1 architecture not being able to generalize.

Model name	1.25e-05	5e-05	0.0001	0.0002	0.0008	0.0032
Reconstruction loss	0.0342	0.0318	0.0361	0.0358	0.3383	0.3384

Table 3.2 – Reconstruction loss on the test dataset for all VAE 1 models

It is also interesting to look at the differences between the energies of the test dataset lattices and their reconstructions, as they are shown in Picture 3.5. The mean total energy of test dataset lattices is $E_{mean} = -26.34$, which means, that the energy differences are very small – even the largest energy difference in Picture 3.5 is less than 0.7 % of the average test dataset lattice energy (not taking into account the VAE 1 0.0032 and VAE 1 0.0008 models).

An assumption we could make is that the energy differences should be positive, meaning the energy of a test dataset lattice should be lower than the energy of its reconstruction. Test dataset skyrmion lattices are a ground state, meaning that they have a minimal energy. However, only a handful of those lattices contain no defects. It could be the inaccurate reconstruction of those defects that causes the energy of the reconstructions to be lower than the energy of the original lattice in some cases.

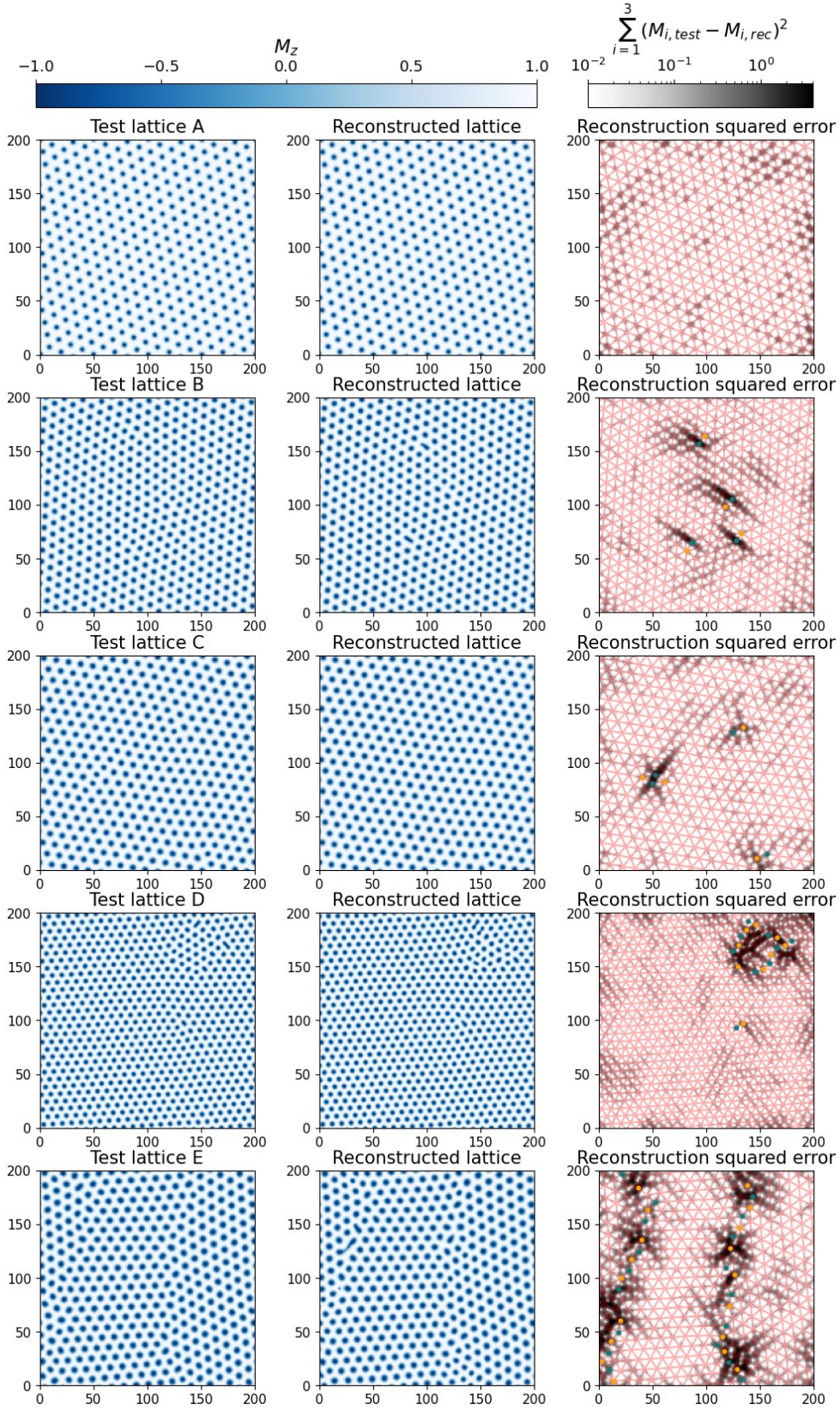
3.5 Examples and analysis of lattice reconstruction

In the previous chapter, we have shown some general metrics illustrating how the VAE 1 models perform on the test dataset, but we did not yet show any specific examples of reconstructed lattices. We will do it here. Since the VAE 1 5e-05 was the model that had the lowest reconstruction loss on the test dataset, we will mainly focus on its reconstructions.

Five examples of lattice reconstructions of the VAE 1 5e-05 model are shown in Picture 3.6. The picture shows a set of test dataset input lattices, their corresponding reconstructions and the squared error between the spins of these test dataset lattices and their reconstructions. The first column shows the M_Z component of the test dataset lattice and the second column shows the M_Z component of its reconstruction. Since the magnetization vector has a unit length, it follows that $M_Z \in \langle -1, 1 \rangle$. In Picture 3.6 in the reconstruction squared error plots, M_1, M_2 and M_3 denote the M_x, M_y and M_z components, respectively. It follows that $\sum_{i=1}^{3} \left(M_{i,test} - M_{i,rec} \right) \in \langle 0, 4 \rangle$.

To better visualise the lattice, the hexagonal pattern was highlighted using Delaunay triangulation. As a set of points for the triangulation were used the centres of skyrmions. As a skyrmion was considered a cluster of at least 7 spins with the component $M_Z < -0.8$. The centre of the skyrmion was chosen as the mean X and Y of this cluster (X and Y in this case denotes the position of the spin on the lattice, not the spin components). Once this was done, we found the skyrmions with either more or less neighbors than six, which are needed to form the hexagonal lattice. These skyrmions form our defects. This can be done by finding how many times a certain node appears in the set of Delaunay triangles. In further chapters, whenever we work with lattice defects, this is the procedure we used to find them.

The lattices from the test dataset were carefully selected to demonstrate a range of scenarios. Initially, we present a lattice free of defects alongside its reconstruction. This is followed by two lattices that contain only a few defects. Finally, we showcase two lattices with a high number of defects. These test dataset lattices were chronologically named A, B, C, D, E.



Picture 3.6 – Visualization of the performance of the VAE 1 model 5e-05 on a few selected lattices (named A, B, C, D, E) from the test dataset. In the first column are the M_z components of lattices from the test dataset that were used as the model's input, in

the second column are the M_z components of the corresponding lattices reconstructed by the model, and in the third column is the squared error between the spins of the test dataset lattice and its reconstruction. Yellow and blue dots mark skyrmions from the input lattice that have more or less than six neighbors, respectively, and the hexagonal structure is highlighted using a transparent red triangulation mesh.

In Picture 3.6, it seems, that the lattices with a higher number of defects have a higher reconstruction loss. This is a claim that is explored in the following chapter. The specifics of how the squared error between the spins of the test dataset lattice and its reconstruction looks around defects is discussed more at the end of this chapter, because further images of lattice reconstructions are shown there. It is also important to note that when displaying the reconstruction squared error, we utilised a logarithmic scale. It helps to increase the contrast of lattices, where the reconstruction squared error is generally very low, such as the lattice A.

Next, we would like to have a look at the correlation between the effective fields and the spins. The effective field is defined as

$$\boldsymbol{h}_{eff,i} = -\frac{\partial H}{\partial \boldsymbol{S_i}} \tag{3.3}$$

where H is the Hamiltonian from the equation (3.1) and $\boldsymbol{S_i}$ is the spin configuration. It plays an important role in the Landau-Lifshitz-Gilbert equation,

$$\frac{d\boldsymbol{S}}{dt} = -\gamma(\boldsymbol{S} \times \boldsymbol{h}_{eff} - \eta \boldsymbol{S} \times \boldsymbol{h}_{eff}), \tag{3.4}$$

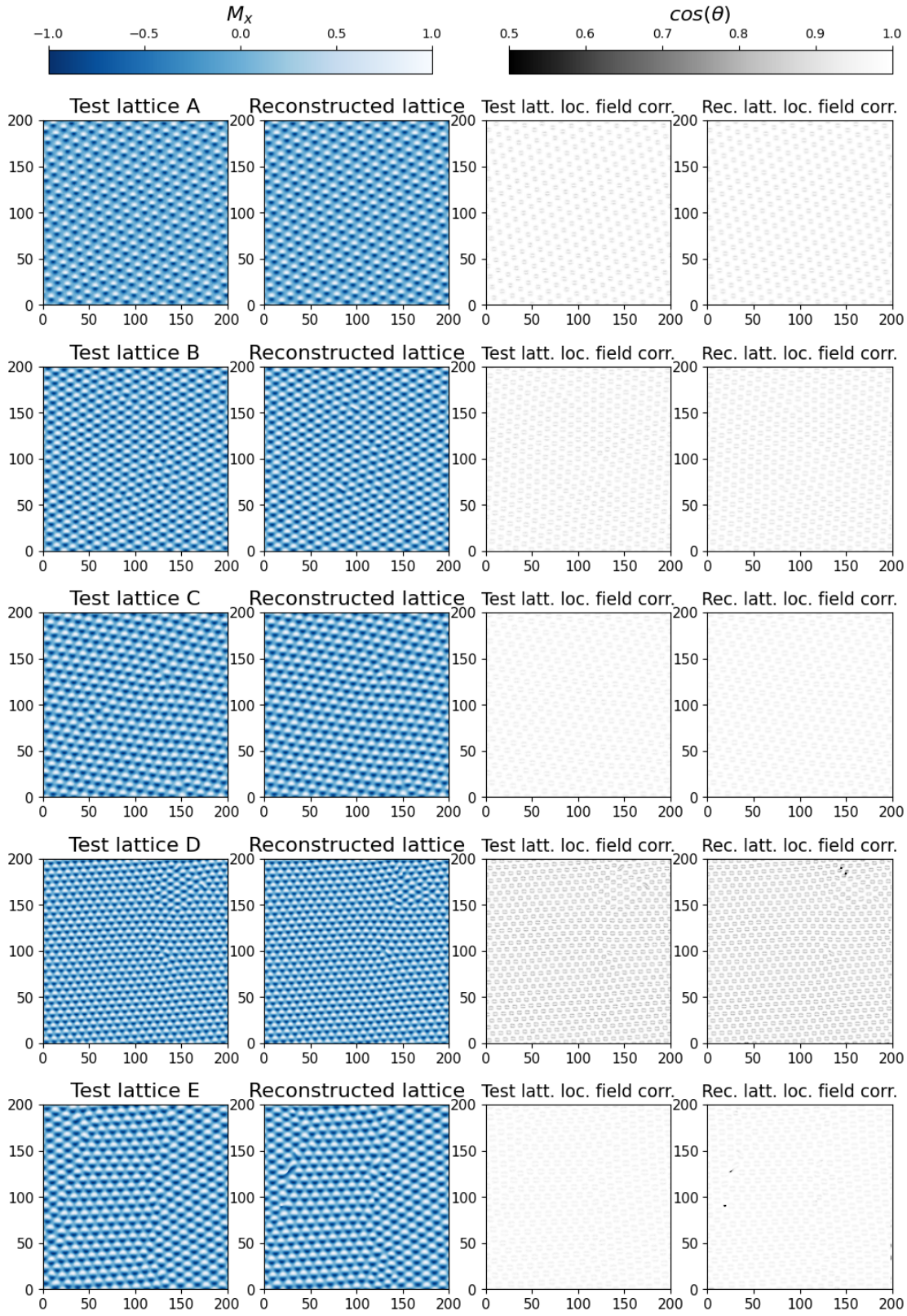
which describes the processional motion in a solid. γ is the electron gyromagnetic ratio and η is the dampening parameter.

The correlation between the spins and the effective fields is then

$$\cos\theta = \frac{\boldsymbol{S}_i \cdot \boldsymbol{h}_{eff,i}}{|\boldsymbol{S}_i| \cdot |\boldsymbol{h}_{eff,i}|} \tag{3.5}$$

We can use this to determine how stable the spin configuration is, because the smaller the angle θ between the spin and the corresponding effective field, the lower the energy [32].

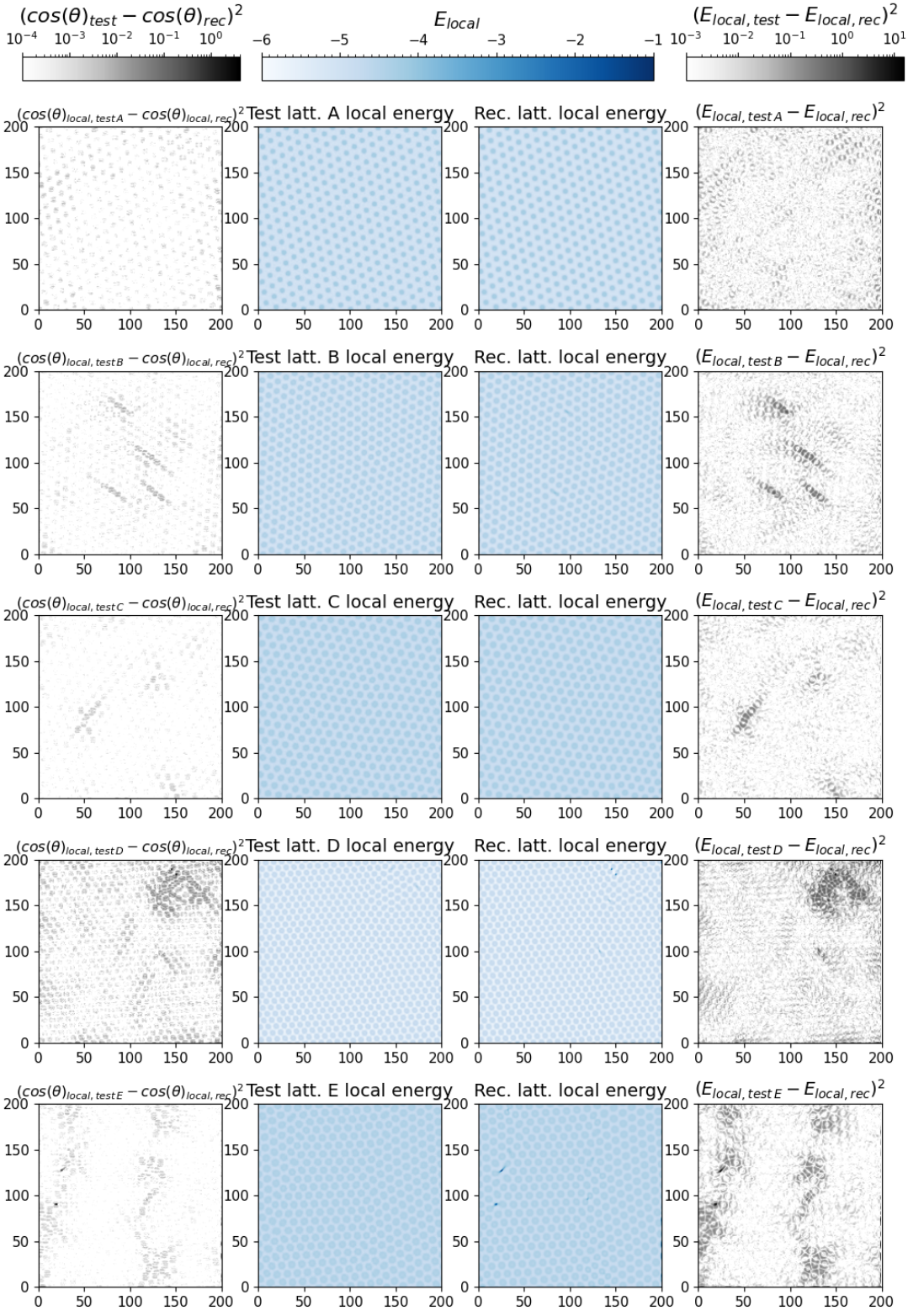
In Picture 3.7, the correlations from the equation (3.5) of the A, B, C, D, E lattices from Picture 3.6 are shown, along with the x components of the magnetization $\boldsymbol{M}$.



Picture 3.7 – Visualisation of the performance of the VAE 1 5e-05 model on the selected test dataset lattices (A, B, C, D, E). The first two columns show the M_x component of the test dataset lattice and the appropriate model reconstruction, respectively. In the third and fourth column is shown the $\cos\theta$ correlation between the

magnetization spins and the effective field $\boldsymbol{h}_{eff}$ of both the test dataset lattice and the reconstruction, respectively.

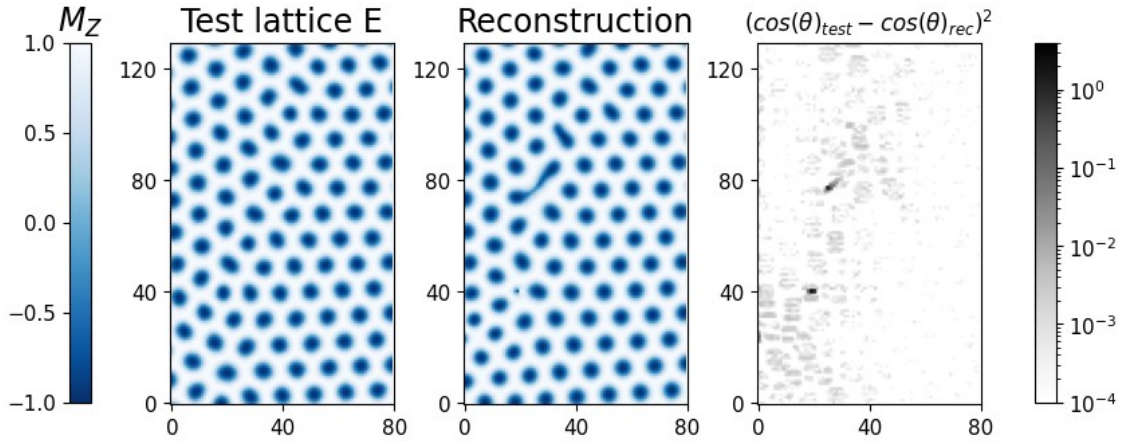
While in lattices A, B, C, the correlation $\cos\theta$ of both the original lattice and the reconstruction are almost the same, lattices D and E have a few spots, where it differs dramatically. To be able to properly compare it, however, requires plotting the squared error between $\cos\theta$ of the test dataset lattice and its reconstruction. This is shown in Picture 3.8 – that way, we can compare the correlations more easily. The local energies of the lattices and their reconstructions, along with the squared error between the two, are also shown there.



Picture 3.8 – Visualisation of the performance of the VAE 1 5e-05 model on the selected test dataset lattices (A, B, C, D, E). The first column shows the squared error between the correlations $\cos\theta$ of the test dataset lattice and its reconstruction, the second and the third column show the local energy of the lattice and its reconstruction,

respectively, and the fourth column shows the squared error between these local energies

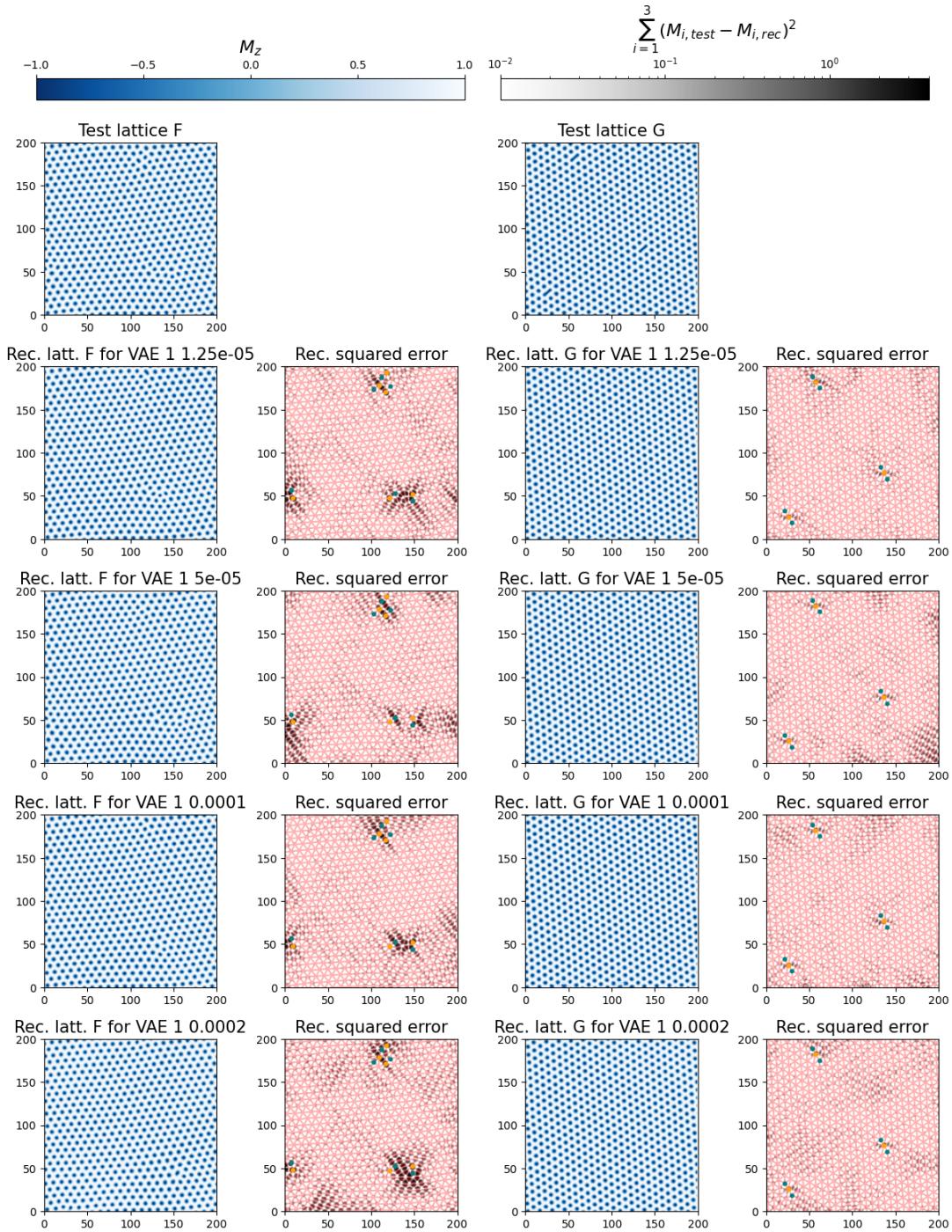
When examining Picture 3.8, it is important to note, that the squared error between the correlations $\cos\theta$ of the test lattices and their reconstructions, and the squared error between local energies, is displayed using a logarithmic scale. As was possible to deduce from Picture 3.7, while the correlations between the lattice and its reconstruction differ minimally for most lattices, there are two points on the lattice E and one point on the lattice D, where the difference is enormous. To better understand it, the problematic region on the reconstruction of the lattice E is shown in Picture 3.9.



Picture 3.9 – The lattice E region of interest. The first plot shows the M_Z component of the test dataset lattice E, the second plot shows the M_Z component of the reconstruction of the lattice E, done using the VAE 1 5e-05 model, and the third plot shows the squared error between the $\cos\theta$ correlations of the said lattice and its reconstruction.

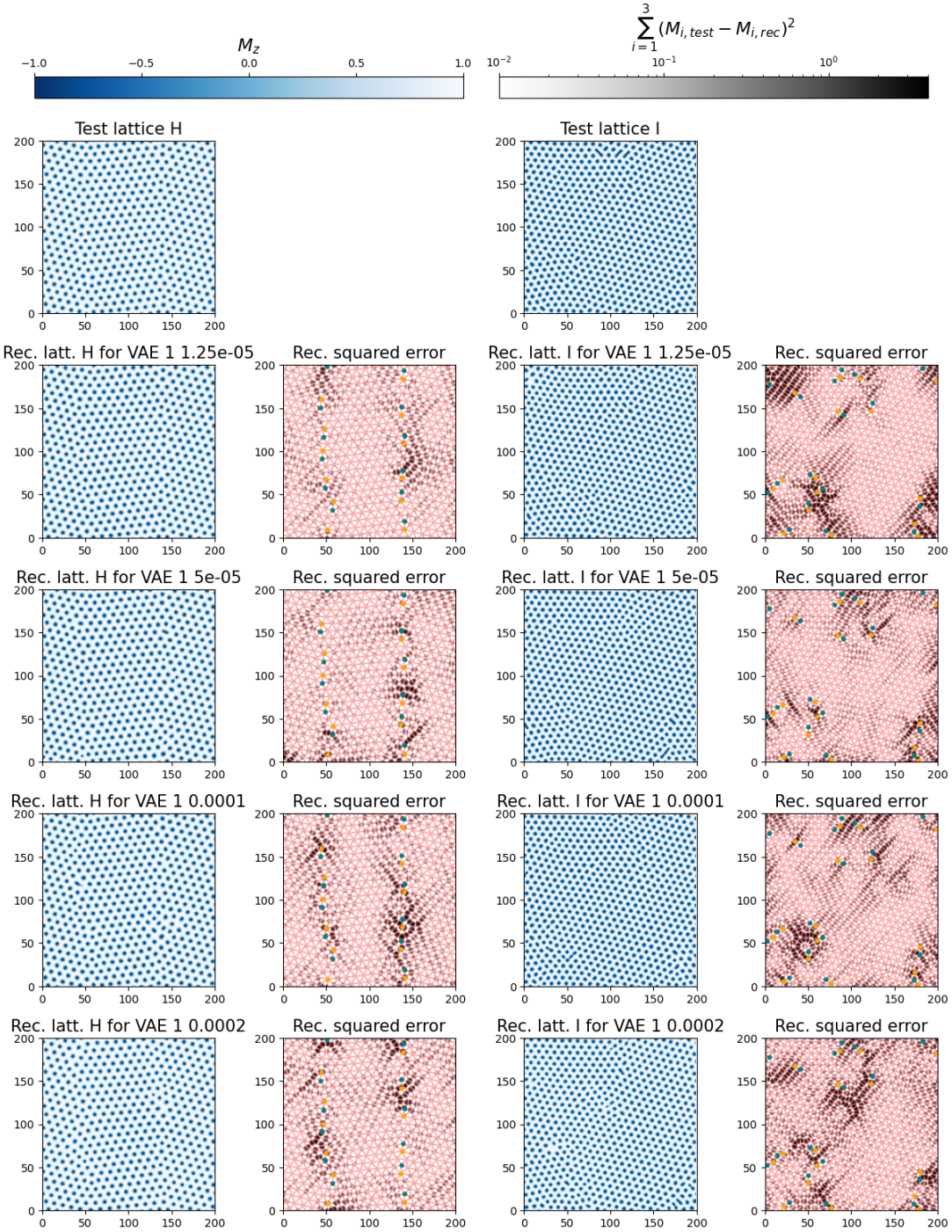
Now, it is easy to see, that those two points in the $(\cos\theta_{test} - \cos\theta_{rec})^2$ plot in Picture 3.9 correspond to the two non-physical artifacts in the reconstruction of the lattice E. It would seem, that the squared error between the $\cos\theta$ correlations of the reference lattice and the reconstructed lattice, or possibly even just the $\cos\theta$ correlation of the reconstructed lattice alone, offers a good way to find non-physical artifacts in the lattices produced by the decoder. In future works, it would be interesting to see whether using the equation (3.5) in the loss function would reduce the number of similar artifacts generated by the variational autoencoder.

We have examined how the selected VAE 1 5e-05 model reconstructs the selected test dataset lattices. Now, let's have a look at a few more lattices, and this time, let's visually compare how the reconstructions differ between all VAE 1 models. That is shown in Picture 3.10 and Picture 3.11.



Picture 3.10 – Visualization of how different VAE 1 models reconstruct test dataset lattices F and G. In the first column in the first row is shown the test dataset lattice F (its M_Z component) and in the remaining rows are its appropriate reconstructions by

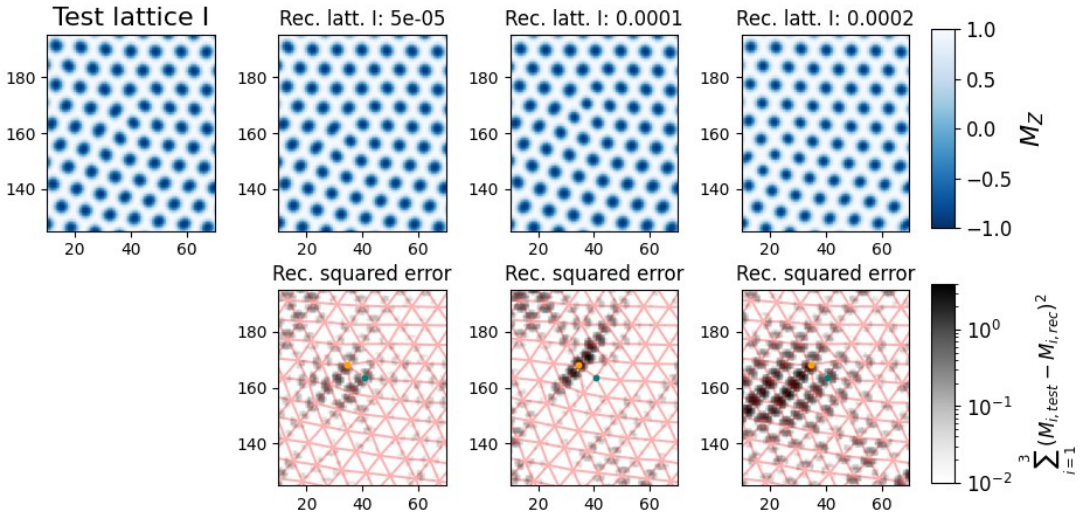
all VAE 1 models. In the second column are the corresponding squared errors between those reconstructions and the test dataset lattice. Analogously, the test dataset lattice G, its reconstructions and the corresponding squared error are shown in the third and the fourth column. Yellow and blue dots mark skyrmions from the input lattice that have more or less than six neighbors, respectively, and the hexagonal structure is highlighted using a transparent red triangulation mesh.



Picture 3.11 –Visualisation of how different VAE 1 models reconstruct test dataset lattices H and I. In the first column in the first row is shown the test dataset lattice H (its M_Z component) and in the remaining rows are its appropriate reconstructions by all VAE 1 models. In the second column are the corresponding squared errors between these reconstructions and the test dataset lattice. Analogously, the test dataset lattice I, its reconstructions and the corresponding squared error are shown in the third and the fourth column. Yellow and blue dots mark skyrmions from the input lattice that have

more or less than six neighbors, respectively, and the hexagonal structure is highlighted using a transparent red triangulation mesh.

Generally, we can see that while all models do a good job reconstructing a regular hexagonal lattice, they struggle with reconstructing lattice defects. Thus, a higher squared error corresponds to those lattice sites. While they sometimes reconstruct a simple 5-7 defects (a defect consisting of one skyrmion with 5 neighbors and one skyrmion with 7 neighbors) perfectly, it is unusual for them to reconstruct more complicated defects without any increase in the squared error. It is not impossible – an example of a very good reconstruction of a complicated defect is the reconstruction of the lattice H by the 1.24e-05 model in Picture 3.11 – but it seems to be improbable.
Some defects also disrupt a large area of the regular lattice structure around it. This leads to the fact, that if a model reconstructs this defect incorrectly, this disruption in the hexagonal lattice around it is incorrect as well, resulting in a high reconstruction squared error in the whole area. A good example of that is the top left corner of the lattice I reconstructed by VAE 1 0.0002 in Picture 3.11. Sometimes, the reconstruction squared error is higher only along the single line of skyrmions the defect is a part of. An example of that is the same region, but this time reconstructed by VAE 1 0.0001. A detail of this part of the lattice is shown in Picture 3.12. A much better reconstruction of the defect by VAE 1 5e-05 model is included.



Picture 3.12 – Visualisation of how selected VAE 1 models (5e-05, 0.0001, 0.0002) reconstruct a certain defect of the test dataset lattice I. The first row shows the M_Z component of the lattice and then their reconstructions using those models, and the

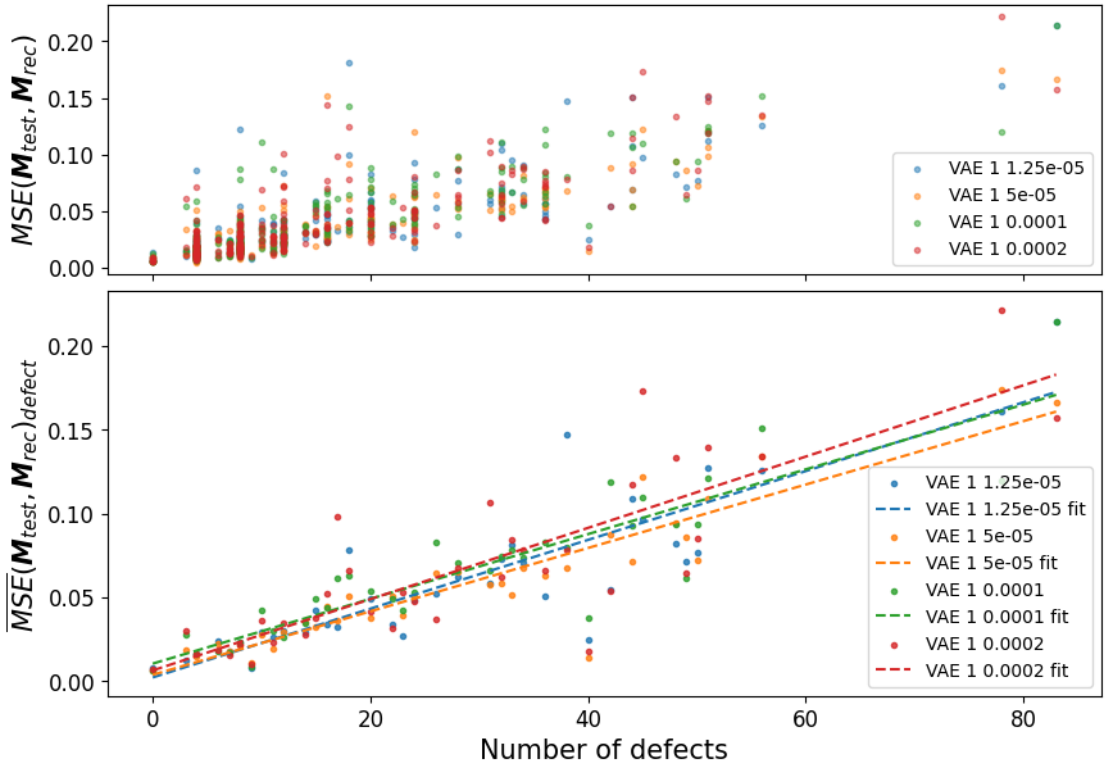
second row shows the reconstruction squared error between the reconstructions above and the test lattice I. Yellow and blue dots mark skyrmions from the input lattice that have more or less than six neighbors, respectively, and the hexagonal structure is highlighted using a transparent red triangulation mesh.

3.6 Reconstruction of lattices with defects

In the previous chapter we have introduced the idea, that the mean squared error between a test dataset lattice and its reconstruction increases with the number of defects in the test dataset lattice. This is what we would now like to explore further.

To do so, we have calculated the number of defects in all test dataset lattices. This was done using the Delaunay triangulation that is described in the chapter 3.4. Once this is done, we can calculate the reconstruction loss of every test dataset lattice. As a defect is considered a skyrmion that has more or less than six neighbors.

We have done this for all VAE 1 models and plotted the dependence of the reconstruction loss of those lattices on the number of defects in those lattices. The result is shown in Picture 3.13.



Picture 3.13 – The first graph shows the mean squared error between test dataset lattices and their reconstructions for selected VAE 1 models dependent on the number of defects of the given test lattice. In the second graphs, these mean squared errors of

lattices with the same number of defects were averaged. Their dependence on the number of defects is displayed along with a linear fit.

We see that the reconstruction loss increases with the number of defects for all VAE 1 models. To make sure that these values truly correlate, Pearson correlation coefficients between the number of defects on a lattice and the average MSE between test lattices and their reconstructions for the same number of defects were calculated for all models. The results are shown in Table 3.3.

VAE 1 model name	1.25e-05	5e-05	0.0001	0.0002
$\rho_{\#def,\overline{MSE(x_{test},x_{rec})}_{def,model}}$	0.88	0.92	0.9	0.84

Table 3.3 – Pearson correlation coefficients between the mean squared reconstruction error of test lattices with the same number of defects and the number of defects in the test dataset lattice for all VAE 1 models.

Pearson correlation coefficients with the value above 0.8 indeed suggest that there is a strong correlation.

3.7 Analyzing variational autoencoder latent space

Besides reconstructions, it is also important to examine the latent space of those models. The latent space is the space of the 500-dimensional vector between the encoder and decoder parts of the model, as shown in Picture 3.3, given by $\boldsymbol{\mu_x}$. $\boldsymbol{\sigma_x}$ gives us variances of those dimensions, it will not be needed in this chapter. The encoder condenses the information about input lattices into this latent space, which is then used by the decoder during reconstruction. We are now interested in this information contained in the latent space.

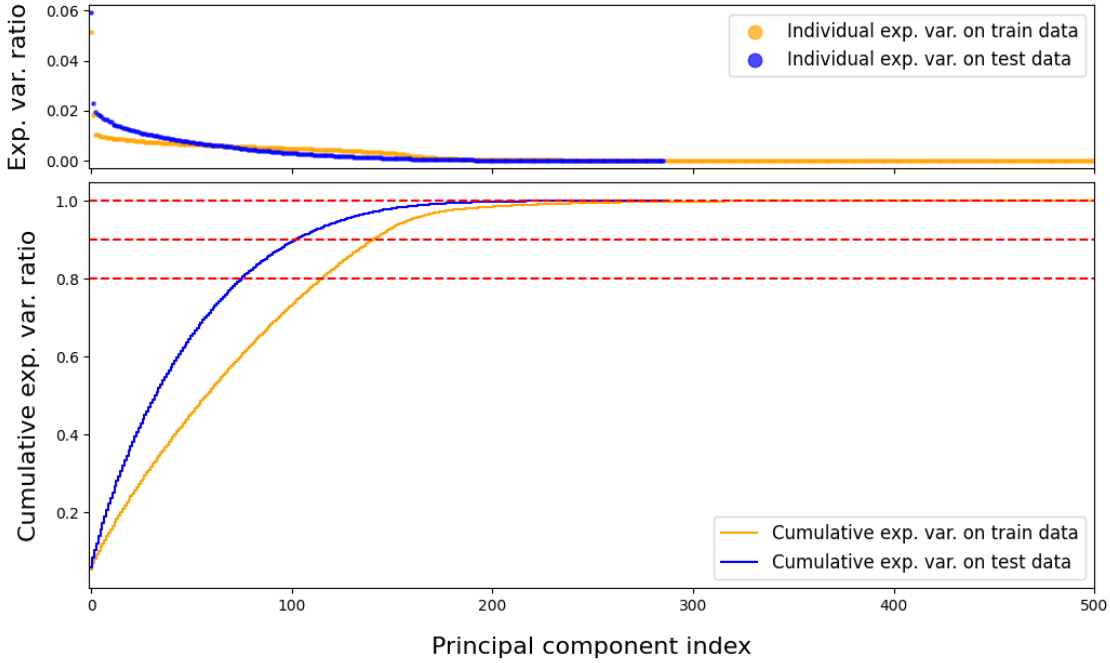
This time, for all test dataset lattices $\boldsymbol{x_1}, \ldots, \boldsymbol{x_N}$, we generated their corresponding latent space representations, meaning $\boldsymbol{\mathcal{Z}_{x_i,\varepsilon=0}} = \boldsymbol{\mu_{x_i}}$, according to the equation (2.20). Then, we applied the principal component analysis on all test dataset $\boldsymbol{\mathcal{Z}}$'s and transformed them using PCA to obtain their corresponding PCA-transformed representations, denoted as $\widehat{\boldsymbol{\mathcal{Z}}}$. This was done for all VAE 1 models independently.

Principal component analysis is needed, because we do not know what information is contained in which latent space dimension (or a set of dimensions). PCA linearly transforms this space so that the elements of $\widehat{\boldsymbol{\mathcal{Z}}}$, called the principal components, are

ordered in such a way, that the first component explains most of the original variance of $\mathcal{Z}$ and the subsequent components explain progressively less variance. This is shown in the equation (2.25).

Looking at how much variance is explained by each component allows us to see how important are the first principal components and how many dimensions we can leave out and still retain most of the information. We can do that using the explained variance ratio and the cumulative explained variance ratio. Explained variance ratio is the explained variance of each principal component plotted for all principal components, and the cumulative explained variance ratio is the summed explained variance of all principal components up until the *n*-th one plotted for all principal components.

This was done in Picture 3.14 for the VAE 1 5e-05 model. Variances for both train and test dataset lattices are shown. Since there were only 286 test dataset lattices, the explained variance ratio for them is shown only for the first 286 principal components. With its 2574 lattices, the train dataset can show the explained variance ratio for the whole latent space.



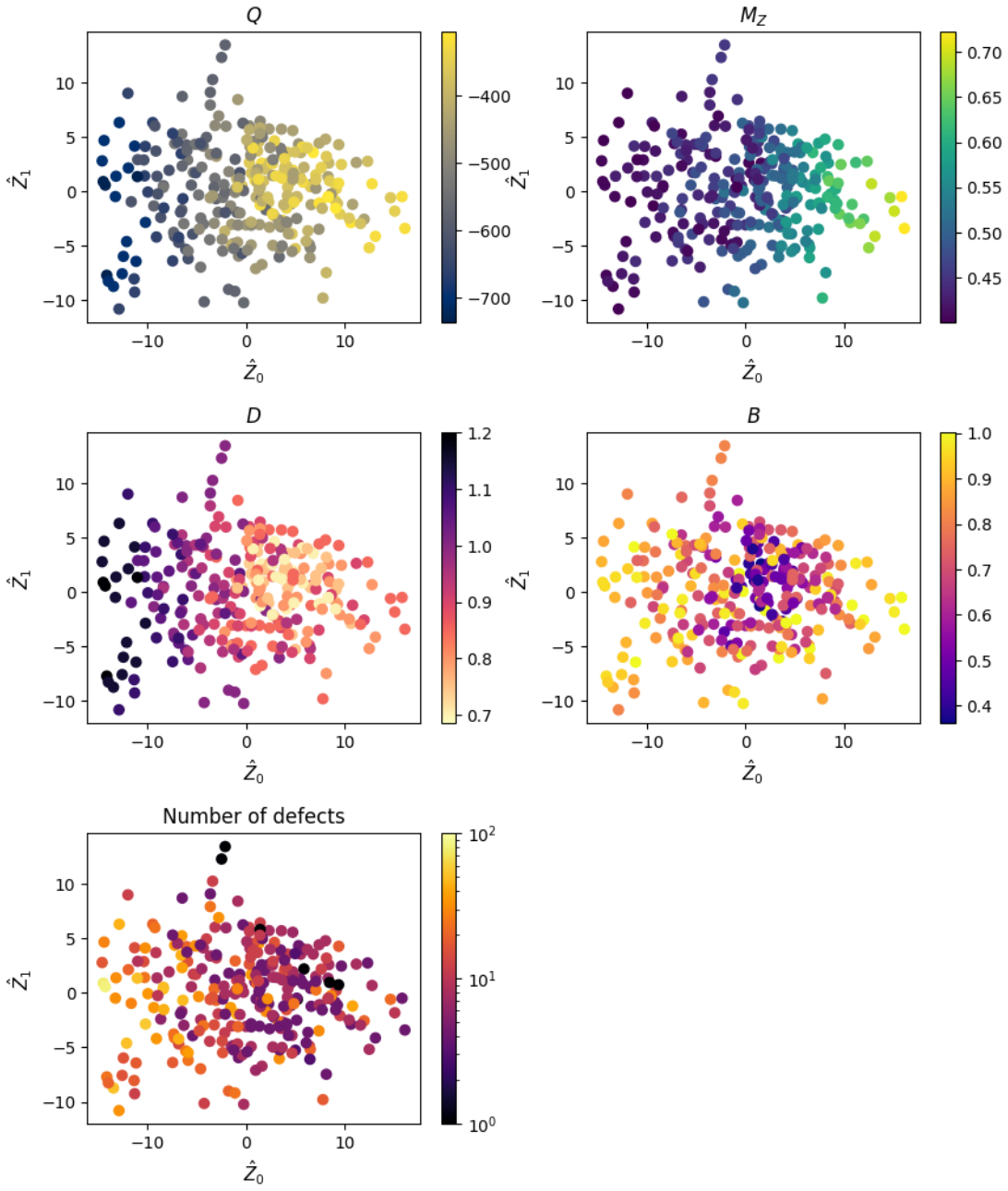
Picture 3.14 – Explained variance ratio and cumulative explained variance ratio of principal components of $\mathcal{Z}$'s for lattices from the train dataset and test dataset respectively, using the VAE 1 5e-05 model

We can see that the first principal component explains substantially more variance than the remaining principal components both on the test dataset and on the train dataset.

The second principal component also has a noticeably larger explained variance. The explained variance of principal components for the test dataset lattices decreases with an increasing principal component index more sharply than for the train dataset – it seems that some components of the latent space learned to find certain features of the train dataset lattices, which were not-so-common in the test dataset (or the model did worse at identifying them). Also, given that the cumulative explained variance ratio reaches 0.999 at the 230th principal component on the test dataset and at the 342nd principal component on the train dataset, the latent space could probably have 100 or even more dimensions fewer.

Next, let's look at the dependence of various quantities of test dataset lattices on the value of the first two principal components of the VAE 1 5e-05 model. Such plots are shown in Picture 3.15.

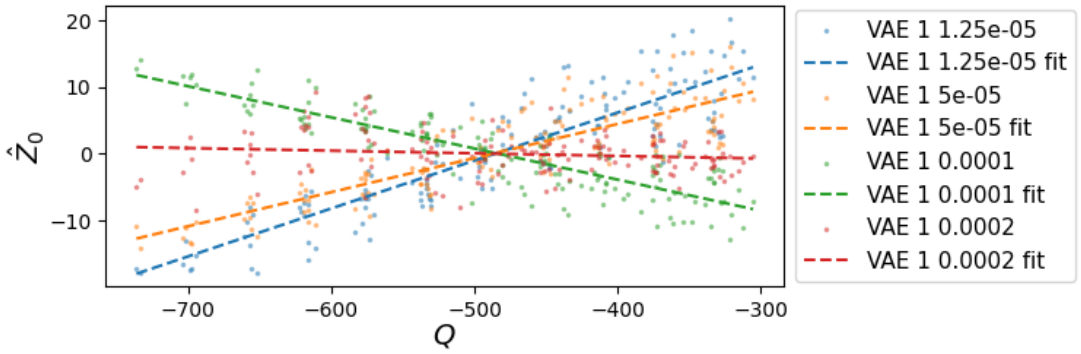
To make the text more concise, the 1st principal component of the latent space $\boldsymbol{\mathcal{Z}}$ is denoted as $\hat{\mathcal{Z}}_0$ and the 2nd principal component of the latent space $\boldsymbol{\mathcal{Z}}$ is denoted as $\hat{\mathcal{Z}}_1$.



Picture 3.15 – The dependence of the size of Dzyaloshinskii–Moriya vector D, the size of the magnetic field B, the average lattice magnetization in the z-axis M_z, the topological charge Q and the number of defects on the first two principal components of latent space representations of the test dataset lattices for the VAE 1 5e-05 model

It is interesting to see that Q, M_Z and D seem to be correlated with $\hat{Z}_0$. To examine whether such correlation occurs even for other VAE 1 models and to better visualise it, $\hat{Z}_0$ of all test dataset lattices with a given Q were averaged and their dependence on

Q was fitted with a linear curve. This was done for all VAE 1 models. The resulting plot is shown in Picture 3.16.



Picture 3.16 – The dependence of the mean $\hat{\mathcal{Z}}_0$ of all test dataset lattices for a given Q on Q for all VAE 1 models, along with a linear fit

Picture 3.16 shows that all $\hat{\mathcal{Z}}_0$'**s** of all VAE 1 models, except for VAE 1 0.0002, correlate with the parameter Q. To find if this is true also about M_Z and D, Pearson correlation coefficients ρ between $\hat{\mathcal{Z}}_0$ and D, Q and M_Z, respectively, were calculated, and they are shown in Table 3.4.

VAE 1 model	1.25e-05	5e-05	0.0001	0.0002
$\rho_{Q,\hat{\mathcal{Z}}_{0,model}}$	0.88	0.85	-0.87	-0.06
$\rho_{M_Z,\hat{\mathcal{Z}}_{0,model}}$	0.85	0.86	-0.83	-0.15
$\rho_{D,\hat{\mathcal{Z}}_{0,model}}$	-0.81	-0.77	0.80	-0.05

Table 3.4 – The correlation between $\hat{\mathcal{Z}}_0$ of test dataset lattices and their Q (second row), mean magnetization in the z-axis M_Z (third row) and D (fourth row) for all VAE 1 models (first row, each column corresponds to a different model)

Table 3.4 indeed shows that all those properties of test dataset lattices correlate with the 1st principal component of their representation in the latent space. The variational autoencoder did not receive any additional information about Q, D or the mean value of the M_Z magnetization of those lattices. It was only trained on the lattices themselves, and this table and graphs demonstrate, that the encoder truly extracts useful and easily interpretable properties of these lattices, such as Q or M_Z, and encodes them into the latent space $\mathcal{Z}$.

However, we need to verify, whether Q, $\bar{M}_Z$ or D of the test dataset lattices are correlated between themselves. Pearson correlation coefficients between these properties of the test dataset lattices are

- $\rho_{D,Q} = -0.98$
- $\rho_{D,\bar{M}_Z} = -0.43$
- $\rho_{\bar{M}_Z,Q} = 0.55$

So, it seems, that the correlation between M_Z and the $\hat{\boldsymbol{\mathcal{Z}}}_0$ is independent on the correlation between D or Q and $\hat{\boldsymbol{\mathcal{Z}}}_0$. However, D and Q appear to be very strongly correlated.

B and the number of defects does not seem to be correlated with $\hat{\boldsymbol{\mathcal{Z}}}_0$. The appropriate correlation coefficients are shown in Table 3.5.

VAE 1 model	1.25e-05	5e-05	0.0001	0.0002
$\rho_{B,\hat{\mathcal{Z}}_{0,model}}$	-0.22	-0.16	0.20	-0.06
$\rho_{\#def,\hat{\mathcal{Z}}_{0,model}}$	-0.50	-0.49	0.52	-0.03

Table 3.5 – The correlation between $\hat{\mathcal{Z}}_0$ of test dataset lattices and their B (second row), number of defects (third row) for all VAE 1 models (first row, each column corresponds to a different model)

Table 3.5 does not necessarily prove that there are no elements of the latent space $\boldsymbol{\mathcal{Z}}$ that try to estimate the magnitude of the magnetic field B or note the number of defects in the lattice. But it certainly is not such a defining characteristic as Q, which correlates with the first principal component of the PCA-transformed $\boldsymbol{\mathcal{Z}}$.

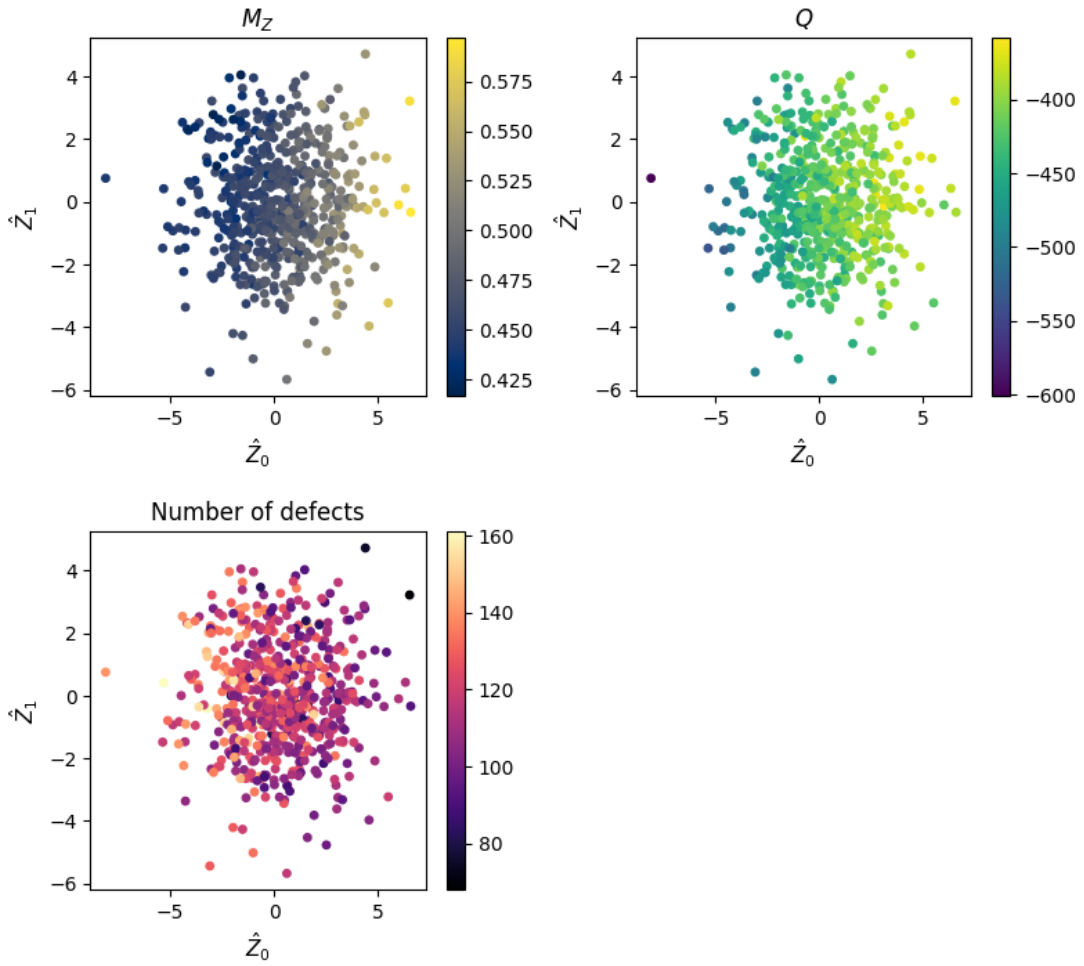
The failure of the VAE 1 0.0002 model to capture the same correlation between the test lattices and its $\hat{\mathcal{Z}}_0$ as the other three models probably means that the KL weight C_{KL} parameter was too high and the model's latent space was being regularized too much. In this context, the failure of the 0.0008 and 0.0032 models with even higher KL weights to train at all is not surprising. However, it is interesting that despite that, VAE 1 0.0002 performed well when reconstructing unseen lattices.

3.8 Generating new lattices

So far, we have thoroughly explored the reconstruction capabilities of the models and the capability of their latent layers to extract important information. Now, let's have a look at their ability to create completely new lattices.

To be able to generate new lattices, we first need to calculate the mean value of $\boldsymbol{\mu}_x$ and $\boldsymbol{\sigma}_x$ as they are shown in Picture 3.3 from the latent representations of all train dataset lattices. We can then sample a random vector $\boldsymbol{\varepsilon}$ from a normal distribution and use the equation (2.20) to generate a new latent vector $\boldsymbol{\mathcal{Z}}$ and then use it as an input for the decoder part of the model, generating a new lattice. Using this procedure, we have created a dataset of 600 new lattices. We transformed the latent vectors corresponding to those lattices using PCA from the chapter 3.7 (see Picture 3.15) to obtain their representation in $\widehat{\boldsymbol{\mathcal{Z}}}$. All was done using the VAE 1 5e-05 model.

First, let's look at whether properties of those generated lattices, such as Q or M_Z, are correlated with $\hat{\mathcal{Z}}_0$. Previously, we have shown that these properties are correlated with $\hat{\mathcal{Z}}_0$ of the reconstructed test dataset lattices. While the model did not see test dataset lattices during training, what if still, they covered only a subspace of the latent space? By examining lattices generated from randomly sampled latent vectors, we can verify that these correlations hold for the whole latent space. The dependence of M_Z, Q and the number of defects of lattices created from these randomly sampled latent vectors using the VAE 1 5e-05 model on $\hat{\mathcal{Z}}_0$ and $\hat{\mathcal{Z}}_1$ is shown in Picture 3.17.



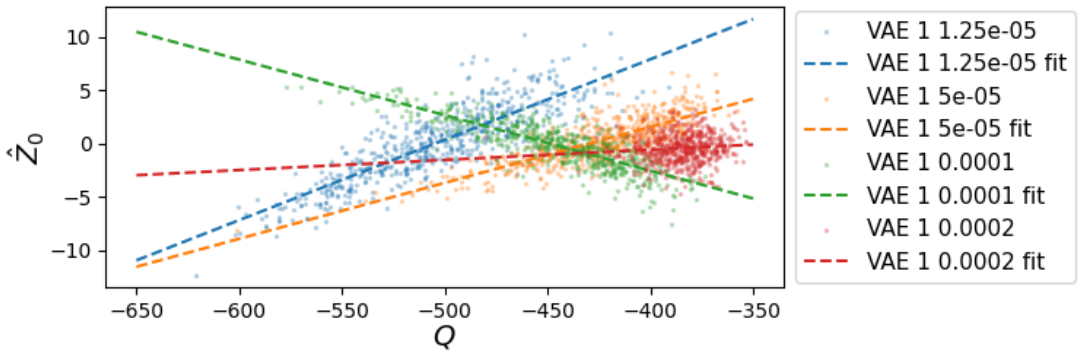
Picture 3.17 – Dependence of M_Z, Q and the number of defects of lattices reconstructed from randomly sampled latent vectors on the first two principal components of those latent vectors transformed using PCA

We see that both Q and M_Z indeed correlate with the first principal component. When we repeated this procedure using the decoder part of other VAE 1 models, we got similar results; M_Z and Q of lattices generated from randomly sampled vectors from their latent spaces also correlated with $\hat{Z}_0$ of their latent representations, except for the VAE 1 0.0002 model. The number of defects does not correlate with $\hat{Z}_0$. The results are shown in Table 3.6.

VAE 1 model	1.25e-05	5e-05	0.0001	0.0002
$\rho_{Q,\hat{Z}_{0,model}}$	0.85	0.79	-0.84	0.09
$\rho_{M_Z,\hat{Z}_{0,model}}$	0.90	0.85	-0.90	0.03
$\rho_{\#def,\hat{Z}_{0,model}}$	-0.05	-0.37	0.01	0.02

Table 3.6 – The correlation between $\hat{Z}_0$ of randomly sampled dataset lattices and their Q (second row), mean magnetization in the z-axis M_Z (third row) and their number of defects (fourth row) for all VAE 1 models (first row, each column corresponds to a different model)

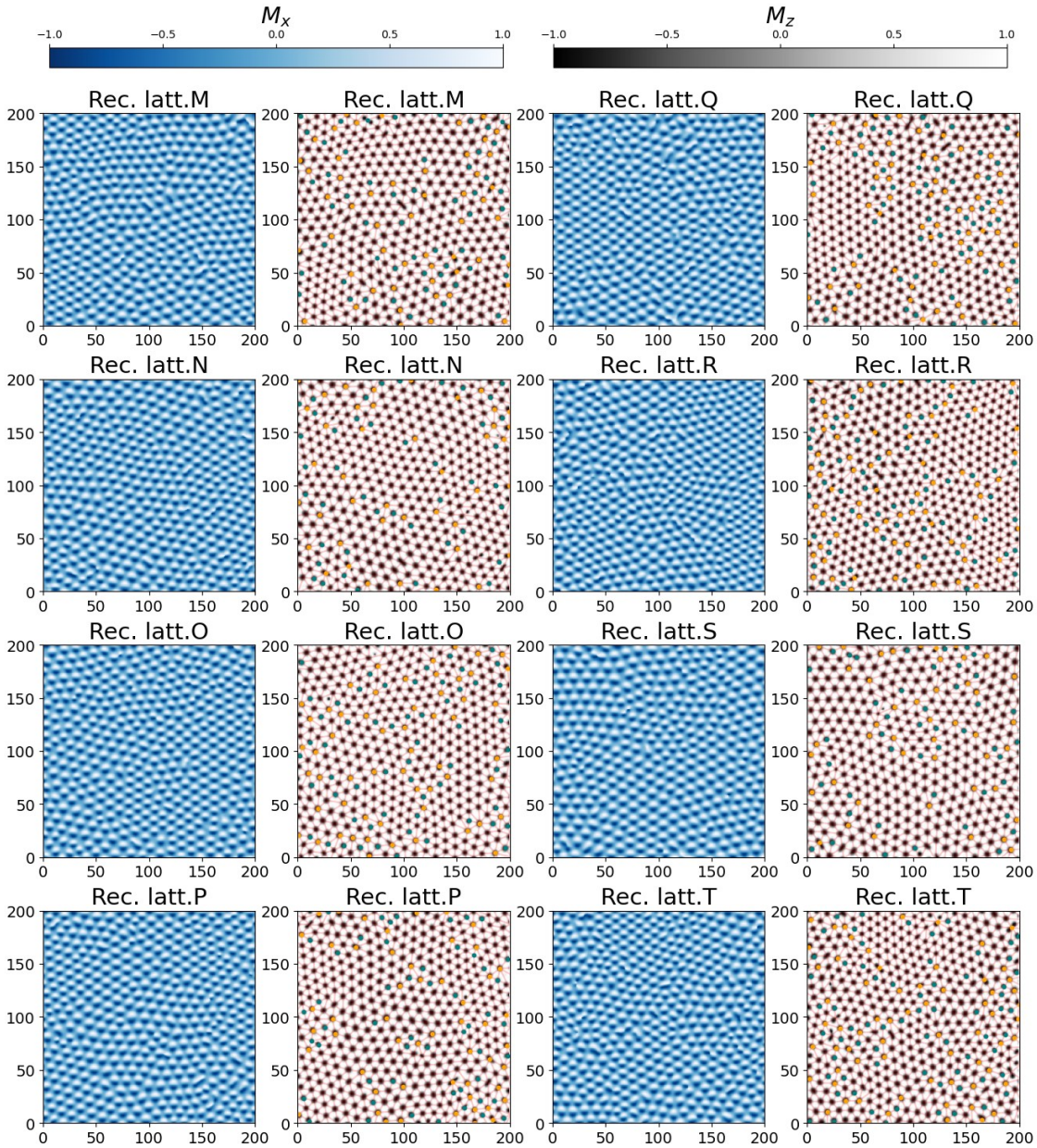
To illustrate data in Table 3.6, the dependence of Q on $\hat{Z}_0$ of these randomly sampled lattices by all VAE 1 models is shown in Picture 3.18.



Picture 3.18 – The dependence of $\hat{Z}_0$ on the topological charge Q of generated lattices by all VAE 1 models from randomly sampled vectors from their latent space $\mathcal{Z}$

At first glance, what is surprising about Picture 3.17 is the generally huge number of defects in generated lattices. These lattices have from 68 to 161 defects, while test dataset lattices had usually an order of magnitude fewer defects (see Picture 3.13). A possible explanation could be that the subspace of the latent space, which contains lattices with none or very few defects, is much smaller than the rest of the latent space. There exist many more possible configurations of lattices with a lot of defects than there are configurations of lattices with no defects. So, when we randomly sample vectors from this space, we are bound generate lattices with a lot of defects. This could be possibly solved by identifying which latent space dimensions are responsible for defects generation by examining latent representations of test and train dataset lattices

with no or only few defects, and sampling random latent space vectors with the constraint that they are in a close proximity of latent space vectors of these lattices. Next, let's have a look at examples of these randomly generated lattices. Eight of these lattices are shown in Picture 3.19.



Picture 3.19 – Examples of lattice generation by the decoder part of the VAE 1 5e-05 model from randomly sampled latent space vectors. Odd columns show the M_x components of these lattices and even columns show their appropriate M_Z components. Even columns also display yellow and blue dots which mark skyrmions from the input lattice that have more or less than six neighbors, respectively, and the hexagonal structure of the lattice is highlighted using a transparent red triangulation mesh.

We see that those lattices truly contain significantly more defects than test dataset lattices. Unlike test dataset lattices defects, which often form boundaries between mostly defect-less regions with a different orientation of the hexagonal pattern, defects in these generated lattices show no such behavior.

Conclusion

In this thesis, we developed a model of a variational autoencoder, trained on a dataset of skyrmion lattices. We examined its ability to reconstruct lattices, to encode the information describing the lattice into a low-dimensional latent space, and its ability to generate new lattices from randomly sampled points in the latent space.

The proposed variational autoencoder architecture features a set of powerful residual blocks and indeed seems to be able to generalize on a dataset of unseen lattices. The architecture was significantly more complex than the one in similar works [32], but that it was needed due to the higher dimensionality of our training data and a significantly higher average topological charge Q of our lattices. Several such models with different weights multiplying the Kullback-Leibler divergence loss during the training were being compared.

When examining the ability of these models to reconstruct lattices, we noticed that they struggled with reconstructing complicated defects and the area around them. This was done by visualizing the reconstruction squared error between the spins of test lattices and their reconstruction. We have shown the dependence of reconstruction mean squared error on the number of defects in those lattices, it appears to be linear and its mean Pearson correlation coefficient for all models is $\rho_{\#def,MSE} = 0.89$. This could be used for the detection of defects in lattices. The squared error between the original and reconstructed spins could be used for the identification of the approximate position of the defect sites.

Furthermore, the latent space of the models was examined. Using the principal component analysis, we have demonstrated that physical properties, such as the topological charge Q or the mean magnetization of the z-axis $\bar{M}_Z$, are indeed being encoded into the latent space, and that they correlate with the first principal component of the latent space. The reduced representation of a lattice in the latent space contains enough information to allow us to reconstruct it with high accuracy.

Lastly, we attempted to generate new lattices from randomly sampled latent space vectors. While the Q and $\bar{M}_Z$ properties of those lattices also correlated with the first

principal component of the latent space, these generated lattices contained a significantly higher number of defects than the test dataset lattices. This was a surprising observation, especially given that the model is generally worse at reconstructing lattices with a high number of defects. Perhaps it is due to the fact, that there are many more possible configurations of lattices with a lot of defects than there are lattices with a low-defect count, and thus random sampling from the space encoding these lattices results in the generation of high-defect count lattices. This would be worth exploring further.

In future works, it would be interesting to encode the information about the magnetic field B and the size of Dzyaloshinskii–Moriya vector D into the model as well. That would allow us to examine the correlation between the effective field and the spins of lattices, or their local energies. Also, one could experiment with the correlation between the effective field and the spins of lattices being a part of the model loss function.

Bibliography

[1] A. Fert, N. Reyren, and V. Cros, “Magnetic skyrmions: advances in physics and potential applications,” *Nat Rev Mater*, vol. 2, no. 7, p. 17031, 2017, doi: 10.1038/natrevmats.2017.31.

[2] B. Göbel, I. Mertig, and O. A. Tretiakov, “Beyond skyrmions: Review and perspectives of alternative magnetic quasiparticles,” *Phys Rep*, vol. 895, p. 1, 2021, doi: 10.1016/j.physrep.2020.10.001.

[3] Karin Everschor-Sitte and Matthias Sitte, “2skyrmions.jpg,” Wikipedia. [Online]. Available: https://commons.wikimedia.org/w/index.php?curid=37682157

[4] S. Li, X. Wang, and T. Rasing, “Magnetic skyrmions: Basic properties and potential applications,” *Interdisciplinary Materials*, vol. 2, no. 2, pp. 260–289, 2023, doi: https://doi.org/10.1002/idm2.12072.

[5] B. Van Dijk, “Skyrmions and the Dzyaloshinskii-Moriya Interaction,” thesis, Utrecht University, 2014. [Online]. Available: http://dspace.library.uu.nl/handle/1874/304188

[6] T. Okubo, S. Chung, and H. Kawamura, “Multiple-qStates and the Skyrmion Lattice of the Triangular-Lattice Heisenberg Antiferromagnet under Magnetic Fields,” *Phys Rev Lett*, vol. 108, no. 1, Jan. 2012, doi: 10.1103/physrevlett.108.017206.

[7] S. Heinze *et al.*, “Spontaneous atomic-scale magnetic skyrmion lattice in two dimensions,” *Nat Phys*, vol. 7, no. 9, pp. 713–718, 2011, doi: 10.1038/nphys2045.

[8] A. Winnacker, “Lattice Defects,” in *The Physics Behind Semiconductor Technology*, Cham: Springer International Publishing, 2022, pp. 81–94. doi: 10.1007/978-3-031-10314-8_6.

[9] C. Kittel, *Introduction to Solid State Physics*, 8th ed. Wiley, 2004. [Online]. Available: http://www.amazon.com/Introduction-Solid-Physics-Charles-Kittel/dp/047141526X/ref=dp_ob_title_bk

[10] L. N. Kanal, “Perceptron,” in *Encyclopedia of Computer Science*, GBR: John Wiley and Sons Ltd., 2003, pp. 1383–1385.

[11] J. Šíma and R. Neruda, *Teoretické otázky neuronových sítí*. Praha: Matfyzpress, 1996.

[12] F. Chollet, *Deep Learning with Python*. Manning, 2017.

[13] K. Hornik, M. Stinchcombe, and H. White, "Multilayer feedforward networks are universal approximators," *Neural Networks*, vol. 2, no. 5, pp. 359–366, 1989, doi: https://doi.org/10.1016/0893-6080(89)90020-8.

[14] Z. Shen, H. Yang, and S. Zhang, "Optimal approximation rate of ReLU networks in terms of width and depth," *J Math Pures Appl*, vol. 157, pp. 101–135, 2022, doi: https://doi.org/10.1016/j.matpur.2021.07.009.

[15] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, "Activation Functions: Comparison of trends in Practice and Research for Deep Learning," *CoRR*, vol. abs/1811.0, 2018.

[16] L. G. C. Evangelista and R. Giusti, "Short-term effects of weight initialization functions in Deep NeuroEvolution," *Evo*, p. 21, 2021.

[17] M. A. Nielsen, *Neural Networks and Deep Learning*. Determination Press, 2015.

[18] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[19] V. Dumoulin and F. Visin, "A guide to convolution arithmetic for deep learning." 2018.

[20] H. Zhang, M. Cisse, Y. N. Dauphin, and D. Lopez-Paz, "mixup: Beyond Empirical Risk Minimization." 2018.

[21] S. Ioffe and C. Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift." 2015.

[22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep Residual Learning for Image Recognition." 2015.

[23] K. He, X. Zhang, S. Ren, and J. Sun, "Identity Mappings in Deep Residual Networks." 2016.

[24] A. Zhang, Z. C. Lipton, M. Li, and A. J. Smola, "Dive into Deep Learning," *arXiv preprint arXiv:2106.11342*, 2021.

[25] M. A. Kramer, "Nonlinear principal component analysis using autoassociative neural networks," *AIChE Journal*, vol. 37, no. 2, pp. 233–243, 1991, doi: https://doi.org/10.1002/aic.690370209.

[26] D. Bank, N. Koenigstein, and R. Giryes, "Autoencoders." 2021.

[27] U. Michelucci, “An Introduction to Autoencoders.” 2022.

[28] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol, “Stacked Denoising Autoencoders: Learning Useful Representations in a Deep Network with a Local Denoising Criterion,” *J. Mach. Learn. Res.*, vol. 11, pp. 3371–3408, Dec. 2010.

[29] L. Pinheiro Cinelli, M. Araújo Marins, E. A. da Silva, and S. Lima Netto, “Variational Autoencoder,” in *Variational Methods for Machine Learning with Applications to Deep Networks*, Cham: Springer International Publishing, 2021, pp. 111–149. doi: 10.1007/978-3-030-70679-1_5.

[30] J. Lever, M. Krzywinski, and N. Altman, “Principal component analysis,” *Nat Methods*, vol. 14, no. 7, pp. 641–642, 2017, doi: 10.1038/nmeth.4346.

[31] O. Eriksson, A. Bergman, L. Bergqvist, and J. Hellsvik, *Atomistic Spin Dynamics: Foundations and Applications*. OUP Oxford, 2017.

[32] H. Y. Kwon, H. G. Yoon, S. M. Park, D. B. Lee, J. W. Choi, and C. Won, “Magnetic State Generation using Hamiltonian Guided Variational Autoencoder with Spin Structure Stabilization,” *Advanced Science*, vol. 8, no. 11, p. 2004795, 2021, doi: https://doi.org/10.1002/advs.202004795.

[33] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization.” 2017.