Exploring deep learning for extracting the features of classical spin systems

Sari Kerckhove

May 31, 2019

FACULTY OF ENGINEERING AND ARCHITECTURE

Exploring deep learning for extracting the features of classical spin systems

Sari Kerckhove
Student number: 01407261

Supervisor: Prof. dr. Jan Ryckebusch
Counsellor: Corneel Casert

Master's dissertation submitted in order to obtain the academic degree of
Master of Science in Engineering Physics

Academic year 2018-2019

GHENT UNIVERSITY

Sari Kerckhove

Exploring deep learning for extracting the features of classical spin systems

May 31, 2019

Reviewers: Prof. Dr. Jan Ryckebusch, Prof. Dr. Wesley De Neve, Corneel Casert and Prof. Dr. Ir. Toon Verstraelen

Supervisors: Prof. Dr. Jan Ryckebusch, Corneel Casert and Tom Vieijra

Ghent University

Theoretical Nuclear and Statistical Physics

Department of Physics and Astronomy

Proeftuinstraat 86, building N3

B-9000 Ghent

Acknowledgement

First and foremost, I would like to thank my promotor professor Ryckebusch. Thank you for your guidance throughout the year and also for the eye-opening course on statistical physics, which sparked my interest in complex systems and machine learning. I would also like to thank my counselors Corneel and Tom, thank you for being always positive, for the proof reading and sharing your experience. Corneel, I could always turn to you with any question on the theory but also on thesis templates and computer problems.

I want to thank my fellow students and friends for making the past years unforgettable. Thanks to the people at INW for the lunches and the frisbee and ping pong breaks.

Fien and Floris, you are fantastic!

Finally, I would like to thank my family. Thank you for always being there for me, for the practical support and for helping me to, more than once, place things in perspective.

Sari Kerckhove May 31, 2019

Abstract

In this work, the application of machine learning for studying features of the Ising model with zero magnetic field is investigated. Three types of machine learning were considered: Firstly, principal component analysis, an unsupervised, dimensionality reducing technique, is used to summarize the high dimensional Ising configurations in a few informative, linear features. Secondly, supervised classification is examined: neural networks are trained to classify Ising configurations in either the ferro- or para-magnetic phase. Finally, neural networks are trained for regression problems, to predict the energy and magnetization of Ising configurations.

It is illustrated that principal component analysis is capable of finding linear order indicating parameters, that allow to locate the phase transition. For classification neural networks, it is shown that the network calculates order and disorder indicating features based upon which the classification is made. Similar to the classical order and disorder indicating features of the Ising model (magnetization and energy) these features vary strongly at the phase transition, resulting in a characteristic error peak. Based on these considerations, it is suspected that there exists a lower limit to the cost function which is independent of the used model. Regression neural networks can be trained to accurately predict energy and magnetization. The prediction of highly disordered states is challenging, but in contrast to classification, a theoretical lower limit on the error was not found.

For each of the three considered techniques, it is demonstrated that the data set used to train the model, should be sufficiently large for good, generalizing performance. Furthermore, it is shown that models, that implement restrictions based on the properties of the posed problem have better performance in general.

Exploring deep learning for extracting the features of classical spin systems

Sari Kerckhove
Promotor: Prof. Dr. Jan Ryckebusch
Supervisors: Corneel Casert, Tom Vieijra

***Abstract*— In this work, the application of machine learning for studying features of the Ising model with zero magnetic field is investigated. Three types of machine learning were considered: Firstly, principal component analysis, an unsupervised, dimensionality reducing technique, is used to summarize the high dimensional Ising configurations in a few informative, linear features. Secondly, supervised classification is examined: neural networks are trained to classify Ising configurations in either the ferro- or para-magnetic phase. Finally, neural networks are trained for regression problems, to predict the energy and magnetization of Ising configurations.**

It is illustrated that principal component analysis is capable of finding linear order indicating parameters, that allow to locate the phase transition. For classification neural networks, it is shown that the network calculates order and disorder indicating features based upon which the classification is made. Similar to the classical order and disorder indicating features of the Ising model (magnetization and energy) these features vary strongly at the phase transition, resulting in a characteristic error peak. Based on these considerations, it is suspected that there exists a lower limit to the cost function which is independent of the used model. Regression neural networks can be trained to accurately predict energy and magnetization. The prediction of highly disordered states is challenging, but in contrast to classification, a theoretical lower limit on the error was not found.

For each of the three considered techniques, it is demonstrated that the data set used to train the model, should be sufficiently large for good, generalizing performance. Furthermore, it is shown that models, that implement restrictions based on the properties of the posed problem have better performance in general.

I. INTRODUCTION

The motivation to study the Ising model through the application of machine learning is explained by the observation that statistical physics and machine learning show a significant similarity: both research domains deal with the extraction of meaningful information and patterns from a large collection of chaotic data [2], [4]. Therefore, machine learning can provide an alternative to the classical mathematical approaches used in statistical physics. E.g. principal component analysis can be used to locate phase transitions through the dimensionality reduction of a data set of unknown configurations [19]. Another example is the training of deep neural networks to learn the energies associated with both short range and long range interactions [15], [16]. This reasoning goes both ways; to gain understanding in machine learning techniques, practising on well understood statistical models such as the Ising model, is a good starting point.

II. ISING MODEL

The two-dimensional square Ising model is a theoretical model of ferromagnetism in statistical physics. The model is described by a set of interacting spins $\{s_i\}$ arranged on an $L \times L$ lattice, which are oriented either up or down ($s_i = +1$ or -1). The energy E of the system[1], results from local coupling interactions in the lattice:

$$E = -J \sum_{i,j=nn(i)} s_i s_j, \tag{1}$$

where J is the interaction strength, $nn(i)$ indicates the nearest neighbors to spin i. For ferromagnetic systems, the interaction strength is positive, $J > 0$, therefore, it is energetically favorable for the spins to be aligned with their neighboring spins. [9]

The Ising model, is a prototypical example of a statistical system that exhibits a second order phase transition. At critical temperature $kT_c \approx 2.269J^2$, the system transitions from the ordered, ferromagnetic phase ($T < T_c$), where most of the spins are aligned, to the disordered, paramagnetic phase ($T > T_c$), consisting of multiple aligned domains with alternating orientations [5], [9], [12]. For infinite lattices, the order parameter, namely the absolute magnetization per spin (figure 1a)

$$\frac{|M|}{N} = |m| = \frac{1}{N} \Big| \sum_{i=1}^{N} s_i \Big|, \tag{2}$$

with $N = L^2$, provides a method to correctly determine the phase of a configuration: $m \neq 0$ indicates ferromagnetism and $m = 0$ indicates paramagnetism.

For finite lattices however, this method does not apply:

- The transition happens less abruptly, spread out over a certain temperature range around T_c, $\Delta T_{m,L}$, which grows broader for decreasing lattice size[3] L (figure 1a).
- Furthermore, the rapid change of the magnetization near the critical temperature $\frac{\partial |m|}{\partial T}\big|_{T \approx T_c}$, goes hand in

[1] in the absence of external magnetic field

[2] k is the Boltzmann constant.

[3] The interpolation of these finite size effects on multiple lattice sizes L, can be used to infer information (in the form of the critical exponents) on the behaviour of the infinite size system [13].

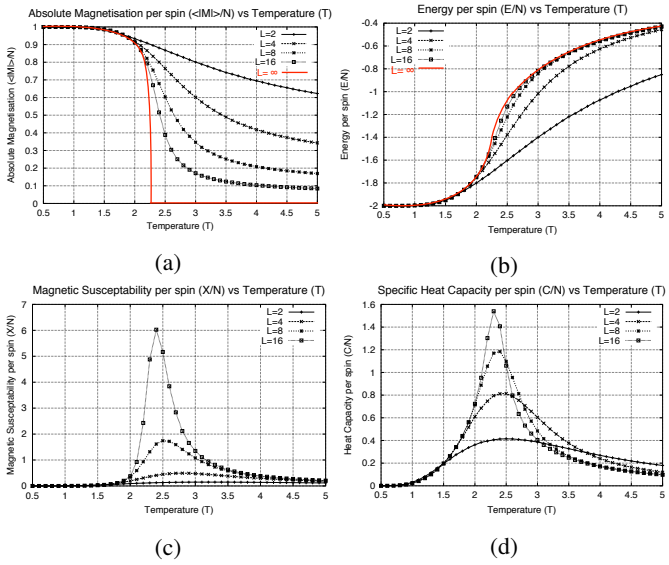


Fig. 1: Properties of the Ising model. In (a) and (b) the magnetization m and energy e per spin are displayed in function of temperature for various lattice sizes. (c) and (d) display the susceptibility χ/N and specific heat capacity C/N per spin, which are proportional to respectively $N\text{var}(|m|)/T$ and $N\text{var}(e)/T^2$, hence they serve as indicators of the fluctuations on m and e. [13]

hand with large variations of the magnetization at the corresponding temperature $\text{var}(|m|)_{T\approx T_c}$ [9].

Therefore, for finite lattices close to transition, $|m|$ can no longer provide certainty on the phase of a configuration. The temperature range $\Delta T_{m,L}$, gives an indication of the area of uncertainty.

In principle, the average energy per spin $e = \frac{E}{N}$ (figure 1b) can also be used to predict the phase to which a certain Ising configuration belongs, e might not be an order parameter in the classic sense of the word [9], still, it can serve as an order indicating parameter. Analogous to the reasoning for $|m|$, for e too, finite lattice size L, goes hand in hand with unreliable phase predictions in a certain temperature range $\Delta T_{e,L}$ around transition.

III. MACHINE LEARNING

In machine learning, a model or function $f(\mathbf{x}|\theta)$ is trained to perform a certain task on a configuration $\mathbf{x}$. The training consists of finding the optimal parameters $\theta*$ for which the model performs best. The performance of a model is always defined in relation to a data set $\mathbf{X}$ of examples, which approaches the configuration space under investigation as well as possible.

A recurring problem in machine learning is the phenomenon of overfitting, which entails that the model performs better on configurations used during training, than on configurations it has not yet encountered. To estimate the severity of overfitting, the data set is divided in a training and testing set. The testing set is not used during the training and solely serves to test if the model has the capacity to perform its task on newly encountered configurations.

Data set

The data set, used in this work, is an assembly of 20 data sets, which, through the application of the Metropolis Monte Carlo algorithm, are each sampled from the configuration spaces accessible to the Ising model at 20 corresponding temperatures surrounding the critical temperature: $T/T_c \in [0.5, ..., 1.5]$, with lattice size $L = 16$ and periodic boundary conditions [9], [16], [17]. All training and testing sets that are used are subsets of this main data set which contains 10^4 configurations per temperature.

Supervised and unsupervised learning

Broadly speaking, machine learning methods can be divided into supervised and unsupervised techniques [11], [14]. In supervised learning, the example configurations are complemented with a label, representing the desired output, which the model learns to predict. This label can either indicate a class to which the configuration belongs (classification), or a continuous property of the configuration (regression). In contrast, in unsupervised learning, it is the algorithm itself that has to find a meaningful way to provide output for each configuration, given the data set of example configurations.

IV. PRINCIPAL COMPONENT ANALYSIS

The unsupervised learning technique, principal component analysis (PCA), is used to find an optimal, linear, low-dimensional representation of the Ising configurations. The original Ising configurations $\mathbf{x}$ are represented in an $N = L^2$ dimensional phase space for which the axes are oriented along the individual spins s_i.

Due to their tendency to align (equation 1), the spins are correlated. The correlations are estimated on the data set $\mathbf{X}$[4] (where each row contains a single configuration), and can be summarized in the covariance matrix:

$$\text{cov}(\mathbf{X}) = \mathbf{X}^T\mathbf{X} \tag{3}$$

The correlations retain redundant information and should be reduced. The orthogonal transformation $\mathbf{A} \in \mathbb{R}^{N\times N}$ that diagonalizes the covariance matrix,

$$\mathbf{A}^T\mathbf{X}^T\mathbf{X}\mathbf{A} = \Lambda \tag{4}$$

with the eigenvalues λ_i sorted in descending order, transforms the phase space such that new axes $\mathbf{A}_{*i} \in \mathbb{R}^{N\times 1}$ will lie along the directions capturing greatest variances in the dataset. The components of each configuration $y_i = PC_i = \mathbf{x}\mathbf{A}_{*i}$, for $i \in \{1, ..., N\}$, expressed in this new basis, are called the principal components. [8], [11], [18]. Retaining only the first, most varying principal components, enables the dimensional reduction of the configurations.

[4]The configurations are assumed to be preprocessed such that they are zero-centered and normalized [1]. For Ising configurations, preprocessing is not necessary because each spin is already zero-centered and normalized.

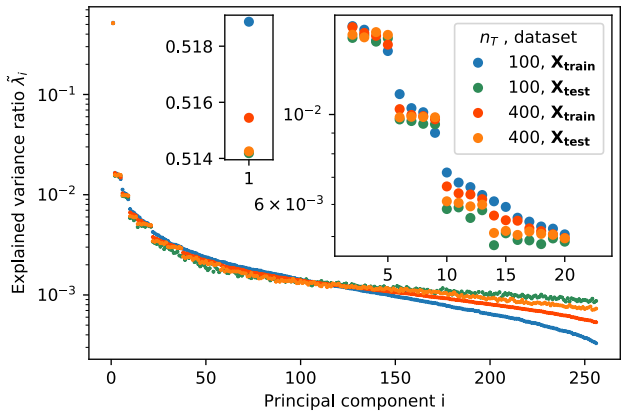


Fig. 2: Results of principal component analysis on Ising configurations. The explained variance ratio $\tilde{\lambda}_i$ for all L^2 principal components is shown for two training sets of sizes 100 and 400 (blue and red curves). The resulting orthogonal transformations $\mathbf{A}_{100}$ and $\mathbf{A}_{400}$ are subsequently used to evaluate the testing set (green and orange curves).

The analysis is done on two training sets, $\mathbf{X}_{train,100}$ and $\mathbf{X}_{train,400}$, containing respectively 100 and 400 configurations per temperature. Delivering two distinct orthogonal transformations $\mathbf{A}_{100}$ and $\mathbf{A}_{400}$. The performance of both transformations was tested on $\mathbf{X}_{test}$ which contains 400 configurations per temperature.

In figure 2, the explained variance ratio

$$\tilde{\lambda}_i = \frac{\lambda_i}{\sum_{j=1}^{N} \lambda_j}, \tag{5}$$

of each of the principle components is displayed. Prominently, there is only one dominant principal component, which already explains half of the variation (51%). All other principal components have far lower variations ($<$ 2%), indicating that a nonlinear function is probably more adequate to describe the second half of the variance.

Figure 3a shows the weights[5], $\mathbf{A}_{*i}$, that determine the corresponding principal components PC_i, for $i \in \{1,2,6,10\}$. The first principal component weighs all spins equally, and is hence equivalent to magnetization m

$$\frac{PC_i}{L} = \frac{1}{N}\sum_{j=1}^{N} s_j = m \tag{6}$$

which is the classical order parameter of the Ising model.

In figure 2, the explained variance ratios of subsequent principal components are ordered in groups. Taking into account the periodic boundary conditions on the configurations, it was found that the weight vectors corresponding to one group are mere translations of one another, proving their physical equivalence.

In ref. [11], the weights $\mathbf{A}_{i*}$ can be identified with the Fourier modes of the spin configuration, which is reflected in figure 3a. The transformation $\mathbf{y} = \mathbf{xA}$ can therefore be understood as the Fourier decomposition of spin configuration $\mathbf{x}$. In the ordered phase ($T < T_c$), the decomposition is dominated by the first principal component (figure 3c): a constant value over the entire configuration. Close to transition, ($T = T_c$), the other principal components, arranged according to decreasing order (increasing frequency f), gain importance (figure 3d). For each principal component, $\langle |PC_i| \rangle$ reaches a maximum at a specific temperature T_i in the paramagnetic phase, T_i could be interpreted as the temperature for which the correlation length $\frac{1}{f_i}$ is dominant.

Figure 3b, shows the projection of the test set onto the two dimensional subspace, spanned by the first two principal components. The configurations are ordered in a centered, high temperature cluster corresponding to the paramagnetic phase and 2 low temperature clusters corresponding to the ferromagnetic phase. From this clustering, it can be concluded that principal component analysis is capable of finding the location of the phase transition. For a temperature range around the critical temperature, some configurations display difficulty to be ascribed to one of the clusters, this reflects the area of uncertainty $\Delta T_{m,L}$, characteristic to finite lattices, for which order parameter $|m|$ displays large variation.

From figure 2, the influence of the training set size can be inferred. In blue and green, the performance of $\mathbf{A}_{100}$ is displayed. The inset shows that the explained variance ratios of the first principal component, $\tilde{\lambda}_1$, is higher for $\mathbf{X}_{train,100}$ then for the $\mathbf{X}_{test}$. The discrepancy between the train and test results indicate that the transformation $\mathbf{A}_{100}$ is overfitted. For $\mathbf{A}_{400}$ (red and orange), the overfitting is strongly reduced. Related, the equivalence of the translated weight vectors, is most visible for $\mathbf{X}_{test}$ followed by $\mathbf{X}_{train,400}$ and least visible for $\mathbf{X}_{train,100}$. The reason is that larger training sets lie closer to the real configuration space, which makes the model less receptive to random noise on the training set.

V. SUPERVISED LEARNING

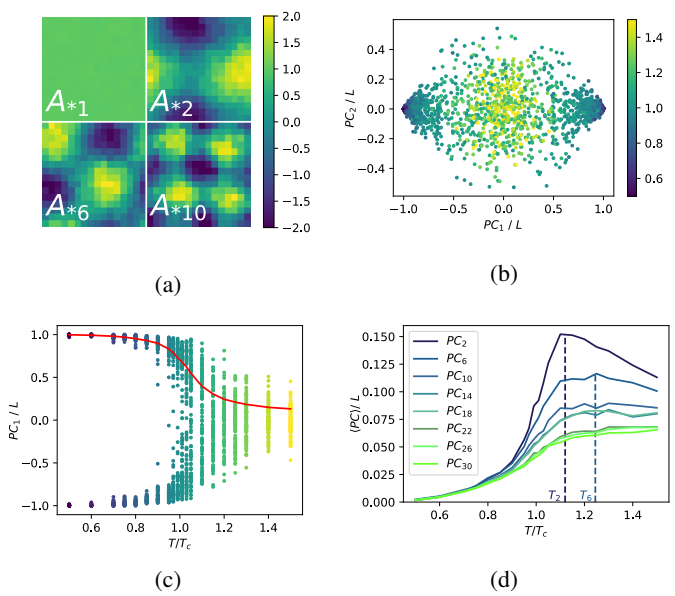
In this section, neural networks will be introduced as a model to perform machine learning. Subsequently, these networks are trained to classify Ising configurations into either the ferro- or the paramagnetic phase. Finally, the same networks are trained to predict the energy and magnetization of Ising configurations through regression.

Neural networks

Neural networks (NN) are a specific type of model, $f(\mathbf{x}|\mathbf{W},\mathbf{b})$, often used in machine learning because of their capability to approach almost any function, provided they have sufficient degrees of freedom[6] (i.e. parameters $\mathbf{W}$ and $\mathbf{b}$). These models are built up out of basic computation units called neurons (figure 4a), which take a weighted sum of a set of input values $\mathbf{wx} + b$, and subsequently evaluate this weighted sum with a nonlinear activation function. The complete network consists of at least two subsequent layers

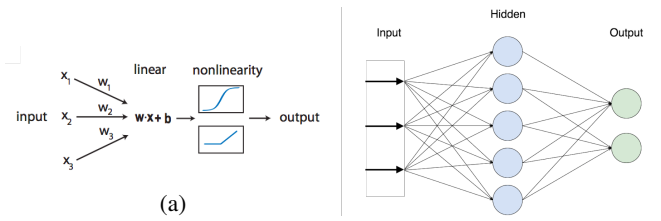
[5] scaled with a factor L

[6] This is the universal approximation theorem [10]



(a) (b) (c) (d)

Fig. 3: Results of principal component analysis on Ising configurations (a) Visualization of the weight vectors corresponding to the first four groups (figure 2) of principal components. (b) Projection of the testing configurations onto the two dimensional subspace spanned by the first two principal components (c) The first principal components of all testing configurations as a function of temperature. In red, $\langle |PC_1| \rangle$ is shown. (d) The absolute average values of higher order principal components $\langle |PC_i| \rangle$ as a function of temperature.



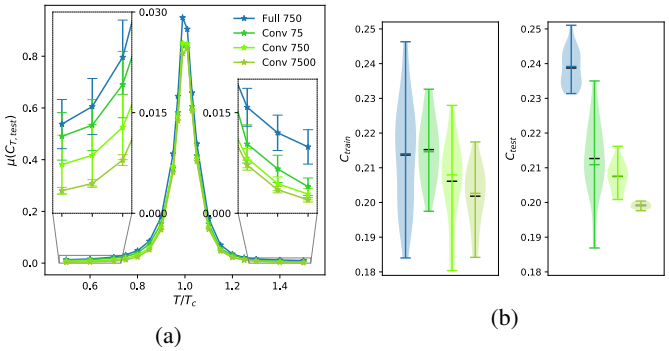
(a) (b)

Fig. 4: (a) the neuron: the basic computation unit of a neural network [14]. (b) Architecture of a neural network [3].

of neurons (figure 4b), each layer takes the outputs of the previous layer as inputs. Two types of network architectures are considered.

1) Fully connected neural networks (FCNN): Without restrictions, all outputs of one layer are used as inputs for each neuron in the next layer (figure 4b). Furthermore, each of these connections is identified with a unique parameter.

2) Convolutional neural networks (CNN): Properties of the input configurations, such as their two dimensional structure and the equivalence of each lattice site are directly incorporated in the model's architecture by imposing restrictions on the degrees of freedom: the majority of connections is discarded, additionally, groups of neurons will share parameters. A single layer of this network type



(a) (b)

Fig. 5: Performance of neural networks on the classification of the Ising configurations (a) average cost on the testing set $\mu(C_{T,test})$ in function of temperature for four types of trained models with differing architecture: fully connected (blue) or convolutional (green) and training set size (shades of green). Each type of training was repeated 25 times. In the insets, the curves are zoomed in and the standard deviation is added. (b) distribution of the total cost on both the training (left) and testing (right) configurations: C_{train} and C_{test}.

corresponds to a set of convolutions. Each convolution can be described as follows: a small kernel of weights, strides over the input, performing the same local analysis, associated with the detection of a certain feature, at every site.

A. Classification

The network $f(\mathbf{x}|\theta)$ takes an Ising configuration $\mathbf{x}$ as input and delivers a single output value y_{pred}, representing the predicted probability[7] that the configuration $\mathbf{x}$ belongs to the ferromagnetic phase. The desired output is represented by a binary label y, which is 1 for a ferromagnetic configuration and 0 for the a paramagnetic configuration. For classification problems, the performance of a model is measured by the cross-entropy cost function [14]:

$$C(\mathbf{x},\theta) = -y\log(y_{pred}) - (1-y)\log(1-y_{pred}) \qquad (7)$$

The total cost on the data set $\mathbf{X}$ equals $\langle C(\mathbf{x},\theta)\rangle_{\mathbf{x}\in\mathbf{X}}$. During training, the optimal parameters dfhaθ^* are found, using a variant of the gradient descent algorithm on the cost of the data set. Due to the large amount of network parameters the efficient backpropagation algorithm is used to calculate the gradients [14].

Results: Figure 5 shows the performance of four types of training runs:

- 1 fully connected neural network (26721 parameters) is trained with a training set containing 750 configurations per temperature.
- 3 convolutional neural networks with the same architecture (771 parameters) are trained with training sets of varying sizes: 75, 750 and 7500 configurations per temperature.

[7] to ensure that $y_{pred} \in [0,1]$ the output neuron contains a sigmoid activation function [14].

Each of these four types of training is repeated 25 times to ensure reliable results. The test set $\mathbf{X}_{test}$ is an assembly of the sets per temperature, $\mathbf{X}_{T,test}$, each containing 250 configurations. In figure 5a, the average cost on the test set $\mu(C_{T,test})$ in function of temperature is displayed for each model. The predominant observation is the characteristic peak in the cost function around the critical temperature T_c. For both model architectures (FCNN and CNN) and the three different sizes of the training set, this peak maintains approximately the same height and width. This observation raises the impression that there is a lower boundary for the cost function.

This suspicion is strengthened by the evaluation of the convolution kernels of the trained CNNs, which are found to calculate order and disorder indicating parameters, based upon which the eventual classification is made. Analogous to respectively magnetization and energy per spin, for finite lattice sizes, these order and disorder indicating parameters display large variations around T_c. Classification based upon these parameters will be uncertain for temperatures in a certain range ΔT around transition, analogous to $\Delta T_{m,L}$ and $\Delta T_{e,L}$, introduced in section II.

As a consequence of this reasoning, it is expected that ΔT will also increase for decreasing lattice size L. This shown in ref. [6], where the cost peaks were subsequently used to perform finite size scaling, inferring information about the infinite model.

Apart from this dominant peak, other results can be obtained. Figure 5b, shows the distribution of the costs for both the training set (left),C_{train}, and testing set (right), C_{test}.

For the fully connected network, the distribution of C_{test} lies significantly higher than the distribution of C_{train}, indicating overfitting. For convolutional networks too, overfitting sometimes occurs, which can be discovered in the low minima of C_{train}, indicating the network's specialization on the training set. However for CNNs the overfitting is far less severe than for the FCNN. Because of the restrictions on the architecture of CNN, significantly less parameters are needed for a model that has similar performance. High dimensional parameterspaces have more freedom to specialize on noise of the training set. In general FCNN performs worse than CNN, comparing C_{test} to C_{train} shows that this is mostly due to overfitting. It can be concluded that models, which implement restrictions based properties of the dataset, have better performance, mainly because less parameters are needed.

The influence of the size of the training set shows better average performance and less variations for larger data sets. This is understandable, because they approach the real configuration space better.

B. Regression

The same network types are now trained to predict either the energy e or the magnetization m per spin of a configuration[8]. For regression, the simplest cost function is the mean squared error [14], [16]:

$$C(\mathbf{x},\theta) = (y - y_{pred})^2. \tag{8}$$

In reality, for finite lattices, e and m are quantized (equations 1 and 2). However, the scale of quantization is small compared to the macroscopic values such that continuous regression can be used. The training method is the same as for classification.

Figures 6 and 7 show the performance of four separate trainings:

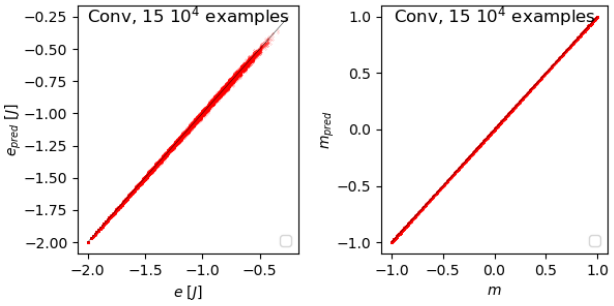
- 3 energy predicting networks: 2 convolutional neural networks, trained with training sets of varying sizes (15 10^3 and 15 10^4, corresponding to 750 and 7500 examples per temperature). 1 fully connected neural network (trained with 15 10^4 examples).
- 1 magnetization predicting convolutional neural network (trained with 15 10^4 examples).

The testing set contains 5 10^4 configurations. The histograms of the data sets in function of energy and magnetization are respectively displayed in figures 7a and 7b.[9]

In figure 6 the predictions of the testing configurations are set out against their true values for both e and m. The results lie close to the diagonal indicating good performance.

In figure 7a, the median of the absolute error $|E_{pred} - E|$ is displayed in function of e for all three trainings and, for every training, both testing and training results are shown. Analogous, figure 7b shows $|M_{pred} - M|$ in function of m.

For all trainings it is found that the error increases for configurations that are increasingly disordered ($e \to 0, m \to 0$). Because of the increasing amount of aligned spin domains and domain borders, the calculation of e and m becomes more challenging. The fluctuations of the curves can be related to the finite number of configurations in each of the bins. For energies exceeding $0.5J$, the predictions are bad and largely fluctuating. The network was insufficiently

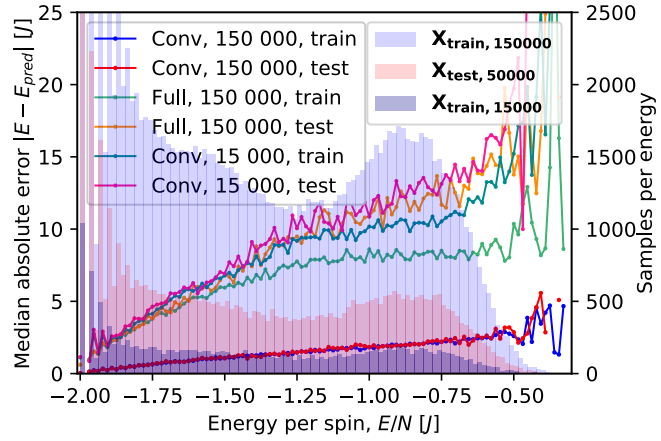


(a) (b)

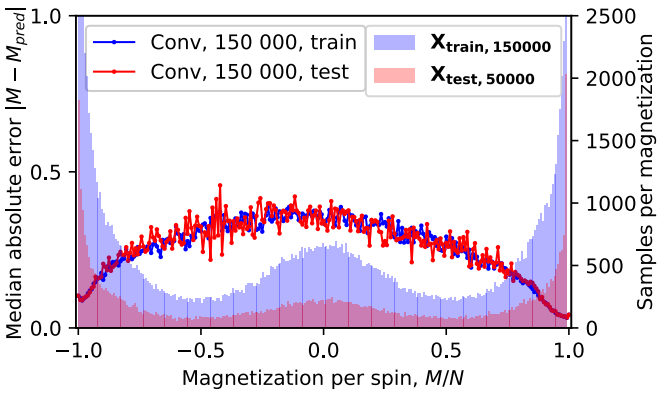
Fig. 6: Performance of neural network for regression. A convolutional model was trained twice to predict e and m, each time a training set of 15 10^4 examples was used. (a) The predicted energy per spin e_{pred} for every Ising configuration in the testing set, is set out against the true energy per spin e. (b) Analogous for m.

[8] the output is no longer a probability, therefore the sigmoid activation of the final neuron is omitted

[9] The histograms have bins that correspond with the real levels of quantization



(a)



(b)

Fig. 7: Performance of neural network for regression. (a) The median absolute error of the predicted energy, $|E - E_{pred}|$ (left axis), is displayed in function of energy per spin e for three distinctly trained models (a FCNN trained on a large training set and 2 CNNs trained on a small and large training set). For each model, the results for both the testing and training configurations are shown. Furthermore, the histograms of the three used data sets are shown (right axis): the large training set (blue), the testing set (red) and the small training set (blue). (b) Analogous representation for the magnetization m of one CNN trained on the large training set.

trained at these energies because they are underrepresented in the training set.

For energy regression, the influence of the size of the training set and the model type can be estimated (figure 7a). The convolutional network, trained with 15 10^4 training samples, has clearly superior performance. Comparing the performance on the training set with the performance on the testing set, shows that overfitting occurs for the convolutional model trained with 15 10^3 examples and also, more severely, for the fully connected model trained with 15 10^4 examples. The convolutional model trained with 15 10^4 examples, is not overfitted on the trained region of energies, but it still fails to generalize on new configurations with other energies. Analogous as for the classification results, we can conclude that the high dimensional parameterspace of the FCNN gives the model more freedom to specialize on noise of the training set. Additionaly, it is found that a larger training set, which is a better approximation of the complete configuration space, is more resistant to overfitting.

The large difference between the errors of the models trained with the the small training set and the large training set, seem to indicate that the cost for regression has no lower limit, in contrast to the characteristic cost peak found for classification.

For the best performing model, the median absolute error for most energies is of the order $2J$, which is smaller than the difference between adjacent Ising energy levels $\Delta E = 4J$. Therefore, we can conclude that for at least more than half of the Ising configurations, rounding off the predictions of the network to the nearest Ising energy level, will give rise to the exact energy of the configuration.

For the magnetization prediction, the median absolute error is lower then 0.5 for all m, which is smaller than the difference between adjacent Ising magnetization levels $\Delta M = 2$. Therefore, we can safely assume that with a few exceptions, rounding off the predictions will give rise to the exact magnetization.

VI. CONCLUSIONS

In this work, it is illustrated that principal component analysis is capable of finding linear order indicating parameters, that allow to locate the phase transition. There is only one dominant principal component, which explains half of the variation (51%). All other principal components have far lower variations, indicating that other non linear unsupervised learning techniques are more adequate to describe the second half of the variance (In reference [6] stochastic neighbor embedding (t-SNE) was used to present a non linear two-dimensional visualization of the Ising configurations.)

For classification neural networks, it is shown that the network calculates order and disorder indicating features based upon which the classification is made. Similar to the classical order and disorder indicating features of the Ising model (magnetization and energy) these features vary strongly at the phase transition, resulting in a characteristic error peak. Based on these considerations, it is suspected that there exists a lower limit to the cost function which is dependent on the physics of the model. In reference [7], it is shown that this error peak can be used for finite size scaling to correctly predict critical exponents of the infinite Ising model.

Convolutional regression neural networks can be trained to accurately predict energy and magnetization. The prediction of highly disordered states is challenging, but in contrast to classification, a theoretical lower limit on the error was not found. The ability of CNNs to learn operators has

been shown to be applicable to various hamiltonians [16], both of short (such as variations of Ising model, e.g. the Potts model) and long range interactions (e.g. the Coulomb hamiltonian), providing accurate results while, in many cases, speeding up calculations enormously [15], [16].

For both classification and regression, it is shown that convolutional neural networks, which implement restrictions in their architecture based on the structure of the Ising configurations, perform better and are less sensitive to overfitting than fully connected neural networks which contain no such restrictions.

For all three types of machine learning, the data set used to train the model, should be sufficiently large to prevent overfitting and to improve performance.

REFERENCES

[1] Everything you did and didn't know about PCA Its Neuronal.

[2] The thing about data. *Nature Physics*, 13(8):717, August 2017.

[3] Artificial neural network, May 2019. Page Version ID: 899336633.

[4] Jeff Byers. The physics of data. *Nature Physics*, 13:718–719, July 2017.

[5] Wei Cai. ME334 Introduction to Statistical Mechanics [Lecture Notes].

[6] Juan Carrasquilla and Roger G. Melko. Machine learning phases of matter. *Nature Physics*, 13(5):431–434, May 2017.

[7] Juan Carrasquilla and Roger G. Melko. Machine learning phases of matter. *Nature Physics*, 13(5):431–434, May 2017. arXiv: 1605.01735.

[8] C. Casert, T. Vieijra, J. Nys, and J. Ryckebusch. Interpretable machine learning for inferring the phase boundaries in a nonequilibrium system. *Physical Review E*, 99(2):023304, February 2019. arXiv: 1807.02468.

[9] H. Gould and J. Tobochnik. *Statistical and Thermal Physics: With Computer Applications*. Princeton University Press, 2010.

[10] Kurt Hornik. Approximation Capabilities of Muitilayer Feedforward Networks. page 7.

[11] Wenjian Hu, Rajiv R. P. Singh, and Richard T. Scalettar. Discovering Phases, Phase Transitions and Crossovers through Unsupervised Machine Learning: A critical examination. *Physical Review E*, 95(6):062122, June 2017. arXiv: 1704.00080.

[12] jwymbs23. Eye-sing Model, January 2018.

[13] Jacques Kotze. Introduction to Monte Carlo methods for an Ising Model of a Ferromagnet. *arXiv:0803.0217 [cond-mat]*, March 2008. arXiv: 0803.0217.

[14] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G. R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to Machine Learning for physicists. *arXiv:1803.08823 [cond-mat, physics:physics, stat]*, March 2018. arXiv: 1803.08823.

[15] Kyle Mills, Kevin Ryczko, Iryna Luchak, Adam Domurad, Chris Beeler, and Isaac Tamblyn. Extensive deep neural networks for transferring small scale learning to large scale systems. *Chemical Science*, 10(15):4129–4140, 2019. arXiv: 1708.06686.

[16] Kyle Mills and Isaac Tamblyn. Deep neural networks for direct, featureless learning through observation: the case of 2d spin models. *Physical Review E*, 97(3):032119, March 2018. arXiv: 1706.09779.

[17] Jos Thijssen. *Computational Physics*. Cambridge University Press, 2 edition, 2007.

[18] Tom Vieijra. Machine learning approaches to strongly correlated spin systems. page 134.

[19] Lei Wang. Discovering Phase Transitions with Unsupervised Learning. *Physical Review B*, 94(19):195105, November 2016. arXiv: 1606.00318.

Contents

1 Introduction

The objective of this work is to investigate the features of the Ising model through the application of machine learning. This approach is motivated by the observation that statistical physics and machine learning show a significant similarity: both research domains deal with the extraction of meaningful information and patterns from a large collection of chaotic data. Therefore, machine learning can provide an alternative to the classical mathematical approaches used in statistical physics. Of course, one can also reverse this reasoning: to gain understanding in machine learning techniques, practising on well-known statistical models is a good starting point. [5, 35]

1.1 Introduction to Machine Learning

Machine learning studies how a model can be trained to perform a certain task, without explicitly programming task-specific rules. The model learns how to perform its task through experience, by considering a large set of examples. [22][25][13]

This approach, in which models are trained for a task without the need for a step by step developed algorithm, makes machine learning very powerful in applications that depend on large and chaotic datasets. Machine learning may not be the most elegant way to analyze this or that dataset, however, because of its generality, numerous different and complex problems can be investigated with the same tools. In the information era, more and more (chaotic) data is stored to be analyzed. This information increase is accompanied with an increase in computer power. This combination results in an increasing interest in machine learning in numerous multidisciplinary fields such as biomedical imaging [2], speech recognition [29], computer vision [18], robotics [34], computational finance [27], psychology [37], etc.

The most common type of machine learning algorithms train models to perform the task of compressing chaotic data. The model is fed an enormous amount of information with the purpose of throwing away the noise and finding out what really matters[1]. E.g. you might feed your model a picture, containing 100 000s of pixels,

[1] This is what the user is interested in. After all, even though one certainly needs to gather data to learn about a certain topic, this gathering is only the first step in the learning process. True learning is not equal to memorizing as much information as possible. A second crucial step in the learning

of either a cat or a dog and it will return you one value, P, which represents the probability that the animal in the picture is a cat. The complete picture is reduced to only one value P. Of course, you can no longer recover the cat's color or the length of its tail, but at least you are almost certain that it is a cat and that's what you're most interested in. A second example, closer connected with the subject of this thesis, is to feed a program an Ising configuration and find out the macroscopic quantities: energy, temperature, magnetization.

We now have an idea of the task a model might perform. The question is how can the model optimally perform this task without being explicitly programmed to do so? How does the model learn?

There are four key components. Starting with a **model** $f(\mathbf{x}|\boldsymbol{\theta})$ and a **dataset** $\mathbf{X}$[2]. A performant model is able to explain or predict certain features of the dataset. To measure the models degree of performance, a **cost function** $C(f(\mathbf{X}|\boldsymbol{\theta}))$ is introduced. The learning consists of finding the optimal parameters $\boldsymbol{\theta}^*$ for which the cost is minimal:

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}}\{C(f(\mathbf{X}|\boldsymbol{\theta}))\}. \tag{1.1}$$

To perform the minimization, a **minimization method** is needed. This method is typically an iterative process[3].

An important matter is overlooked. Indeed, we tried to minimize the cost function C, by using known configurations. However we don't want the model to only perform well on these known configurations. A good model has the capacity to generalize, it should make accurate predictions in every situation, even when it is offered a configuration it has never encountered before. A model that performs worse on new configurations than on the configurations used during training is said to be overfitted.

For this reason, it is interesting to subdivide the dataset $\mathbf{X}$ in a train- and test-set: $\mathbf{X}_{train}$ and $\mathbf{X}_{test}$. $\mathbf{X}_{train}$ is used to actively learn what parameters minimize the cost function. $\mathbf{X}_{test}$ is used to find out if the resulting model has the capacity to generalize and make accurate predictions for unknown configurations.

As noted earlier, the most common types of machine learning, train models to perform the task of compressing chaotic data. Below, four main categories are summarized:

process is to simplify information by finding structures and patterns in order to find out what's important and gain insight.

[2] In this text, the convention is that $\mathbf{X}$ represents a set of configurations, whereas $\mathbf{x}$ represents a single configuration.

[3] There are exceptions, e.g. PCA uses a 1 step orthogonal transformation to minimize the cost.

1. Classification: The model (e.g. a Neural Network) is trained to divide the data into predefined classes.

2. Regression: The model $\tilde{f}$ (e.g. a Neural Network or a polynomial) is trained to approximate a real-life process $f(x)$ that assigns a continuous label y to an observation or datapoint x. A simple example of regression is the fitting of a polynomial.

3. Clustering: Clustering techniques such as K-means aim to find a meaningful division and classification for the input-data. They divide the data into different 'clusters'.

4. Dimensionality Reduction: Dimensionality Reduction techniques such as Principal Component Analysis (PCA) aim to transform high dimensional data to a lower dimensional space while retaining the most important information.

These categories can be ordered in different ways. A first way to classify ML-techniques is by making the division between supervised and unsupervised learning. In supervised learning, the labels or classes, which are to be allocated to the data-configurations, were predefined by the user. In unsupervised learning however, it is the algorithm itself that has to find a meaningful way to label the data. Another way to classify ML-techniques is based upon the character of the output. The data can be represented in a discrete set of labels, classes or clusters on the one hand or a continuous label-range on the other hand. Table 1.1 illustrates these divisions.

Tab. 1.1: machine learning: main categories

ML Tasks	Supervised	Unsupervised
Discrete	Classification: e.g. Neural Networks	Clustering: e.g. K-means
Continuous	Regression: e.g. Function Approximation	Dimensionality Reduction: e.g. Principal Component Analysis (PCA)

Compressing chaotic information is not the only task that can be performed using machine learning. E.g. a self-driving car is expected to drive to a certain destination in compliance with the traffic rules, a general adversarial network (GAN) is trained to generate new, unseen configurations from the probability distribution of the dataset [39].

However, even though these tasks seem to be very different, structuring and compressing chaotic datasets is always an indispensable aspect of the algorithm. A self-driving car relies strongly on the compression of the signals intercepted by its sensors and an important part of a GAN is the discriminator, which is a classifier

that predicts whether a configuration belongs to the original dataset or was newly generated by the network. Therefore, the main categories listed in table 1.1 can be considered as building blocks in more advanced models.

1.2 Introduction to the Ising model

The previous section indicates that statistical models are good candidates to be investigated through machine learning techniques. A model can be trained to find the most important features from a chaotic input-configuration. A possible chaotic input-configuration could entail a set of coordinates and velocities corresponding to atoms in the gaseous phase. For these configurations, a model could be trained to extract thermodynamic state functions such as the energy per degree of freedom e or the temperature T.

Of course, one can also reverse this reasoning: to gain understanding in machine learning techniques, practising on well-known statistical models is a good starting point. The Ising model is a mathematical model that describes magnetism in statistical physics. The two-dimensional square-lattice Ising model is an important and well-studied model because it is the prototype of a statistical system that undergoes a phase transition. A phase transition is characterized by a sudden change in physical properties as a result of a change in external parameters. For the Ising model, the transition between the paramagnetic and the ferromagnetic phase is characterized by a sudden change in the magnetization as a result of changing temperature. Models that undergo a phase transition are well suited for testing machine learning classification algorithms, where configurations are classified according to their phase.

1.2.1 Definition of the Ising model

The square Ising model is described by a set of binary spins arranged on a square lattice. For a system consisting of $N = L \times L$ spins $\{s_i\}$, that can be either up (+1) or down (-1), where index i indicates the position in the lattice, the energy[4] E of the system can be expressed in the form

$$E = -H \sum_{i=1}^{N} s_i - J \sum_{i,j=nn(i)}^{N} s_i s_j. \tag{1.2}$$

[4]Since Ising models can be defined for various lattice sizes N, the energy per degree of freedom gives more insight: $e = \frac{E}{N}$

The first term describes the interaction of every spin with an external magnetic field of strength H. This term causes the spins to align with the magnetic field (to greater or lesser extent depending on the temperature). The second term describes local coupling interactions in the lattice, where J is known as the exchange constant. Only spin pair, nearest neighbour interactions are considered. At low enough temperature, this term causes the spins to align with their neighbouring spins, even in the absence of an external field $(H = 0)$. These local interactions are what make the phase transition, from the paramagnetic to the ferromagnetic phase, possible.

In this thesis, the Ising model will be studied in the absence of an external field $(H = 0)$. The energy now solely consists of the coupling term and the model becomes invariant to a sign-reversal of the spins in all lattice sites. The exchange constant J will be considered positive $(J > 0)$, causing the spins to align in parallel with their neighbouring spins.[5][12][6]

1.2.2 Behaviour of the Ising model

Infinite lattice

In figure 1.1a, the magnetization per spin

$$m = \frac{M}{N} = \frac{1}{N}\sum_{i=1}^{N} s_i \tag{1.3}$$

of an infinite square lattice in function of temperature is shown (in red). The magnetization, m, is the order parameter of the Ising model, at a specific temperature, m changes abruptly, marking the transition between two distinct phases. This temperature is called the critical temperature T_c, for a two-dimensional square lattice, kT_c is found to be $2.269\ J$.[6]

At low temperatures $(T < T_c)$, the spins in the system are all aligned to minimize exchange energy (figure 1.2). This is the ordered, ferromagnetic phase and it has a net magnetization per spin $m \approx +1$ or -1. When the temperature approaches the critical temperature $(T \to T_c)$, it becomes advantageous for the system to display some disorder, because the entropy related term in the free energy

$$F = E - TS, \tag{1.4}$$

[5] This is called the ferromagnetic model. Negative values of J cause the spins to align in anti-parallel with their neighbouring spins. This is called the anti-ferromagnetic model.[6]
[6] where k represents the Boltzmann constant.

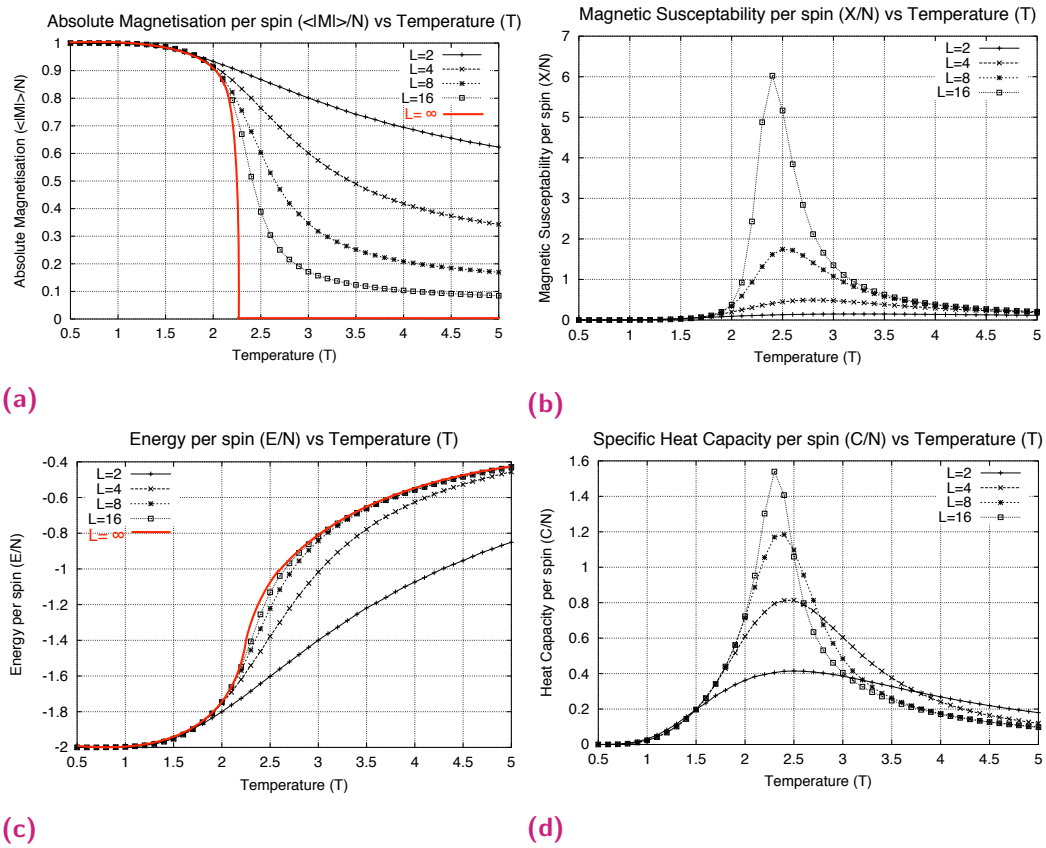


Fig. 1.1: Properties of the Ising model. In (a) and (c) the magnetization m and energy e per spin are displayed in function of temperature for various lattice sizes. (b) and (d) display the susceptibility χ/N and specific heat capacity C/Nk_B per spin, which are proportional to respectively $N(\Delta m)^2/T$ and $N(\Delta e)^2/T^2$, hence they serve as indicators of the fluctuations on m and e. (source figures: [20])

gains importance. Little domains of opposite spins start to occur, getting larger and larger as temperature rises. The net magnetization m decreases.

At the critical temperature ($T = T_c$), the spin-spin correlation length diverges and the spins are organized in aligned domains of strongly varying sizes. Despite the alignment, for infinite systems, the total number of up-spins equals the total number of down-spins and the net-magnetization m becomes zero.

For $T > T_c$, the aligned domains get smaller and the spins become more independent, increasing the entropy of the configuration. This is the disordered, paramagnetic phase.[12][17]

In the theoretical situation of classifying infinite lattice configurations, both m and e can perfectly predict the phase of a configuration (since they show no variations due to the infinite lattice size).

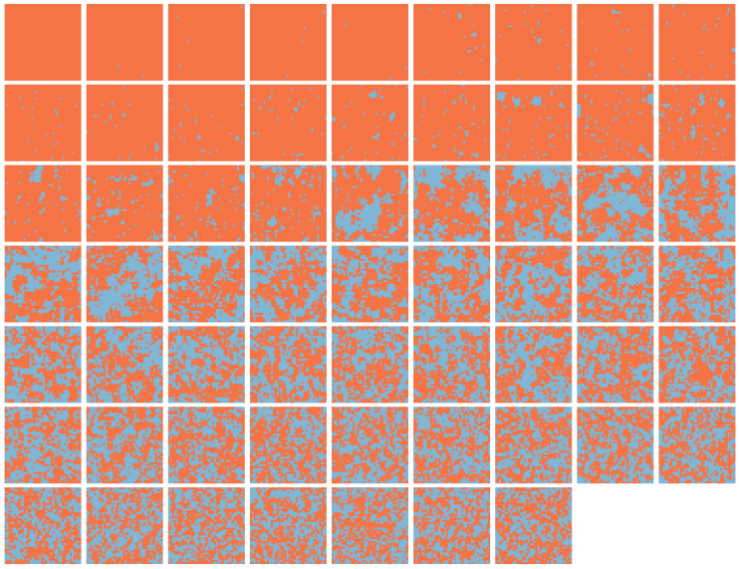


Fig. 1.2: Ising-configurations on a 40×40 lattice, for 61 different temperatures $\frac{kT}{J} \in [1, 5] \Leftrightarrow \frac{T}{T_c} \in [0.44, 2.20]$. At low temperatures ($T < T_c$), the spins in the system are all aligned to minimize exchange energy. As T approaches the T_c, little domains of opposite spins start to occur and the net magnetization m decreases. At the critical temperature ($T = T_c$), the spin-spin correlation length diverges and the spins are organized in aligned domains of strongly varying sizes. For $T > T_c$, there is no net magnetization and the spins show little long range order, the aligned domains get smaller.(source figure: [17])

Finite lattice

The behaviour of the Ising model on a finite lattice differs slightly from the behaviour of the Ising model on an infinite lattice. Figures 1.1a and 1.1c show the magnetization and the energy in function of temperature for an varying lattice sizes.[7] The transition is less abrupt, it is spread out over a small temperature range, situated around T_c which increases for decreasing lattice size L.

At the critical temperature ($T = T_c$), the total number of up-spins no longer equals the total number of down-spins as was the case for the infinite system. On the finite scale, one direction of alignment will be dominant over the other. Therefore, the net magnetization $|m|$ of finite systems will differ from zero. Furthermore, a finite lattice size implies fluctuations of $|m|$ and e at fixed temperatures. This variation can be related to the magnetic susceptibility χ

$$\frac{\partial \langle |M| \rangle}{\partial B}_T = \chi = \frac{\text{var}(|M|)_T}{kT} = \frac{N^2 \text{var}(|m|)_T}{kT}, \tag{1.5}$$

[7] The solution of the Ising model on a finite lattice depends on the boundary conditions. All results shown here, make use of a periodic boundary condition.

and the heat capacity C

$$\frac{\partial \langle E \rangle}{\partial T}_B = C = \frac{\text{var}(E)_B}{kT^2} = \frac{N^2 \text{var}(e)_B}{kT^2}, \tag{1.6}$$

where B represents the netto magnetic field and k represents the Boltzmann constant. [36] The peaks in the susceptibility and heat capacity around transition temperature (figures 4.4c and 1.1d), indicate strong fluctuations on $|m|$ and e for finite lattices. These fluctuations imply that, in contrast to the infinite lattice, for a certain temperature range, ΔT_L, around the critical temperature T_c, $|m|$ and e are no longer capable of providing a correct prediction of the phase of a configuration. This range of uncertainty increases for decreasing lattice size L.

1.2.3 Generating a dataset of Ising configurations

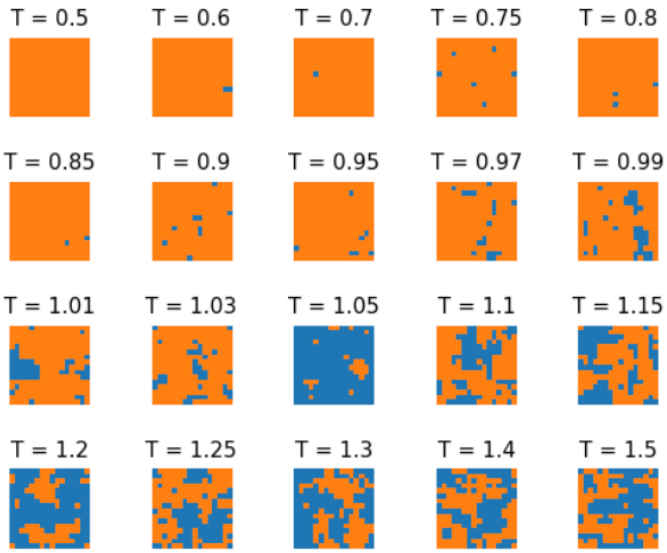


Fig. 1.3: The total dataset is constituted from samples for 20 different temperatures T (expressed in units T_c). For each temperature one configuration is shown. The configurations have size 16×16 with periodic boundary conditions.

The Ising model is a theoretical construct and it has no prescription for its dynamical evolution. Therefore, the Ising model can only be simulated using artificial dynamics based on random numbers (Monte Carlo). In Metropolis Monte Carlo, a chain of configurations is generated for a system in equilibrium with a thermal bath at temperature T (called a Markov chain). One element of the chain is generated by one Monte Carlo step:

- loop over all spins in the lattice:
  - suggest a spin-flip

- determine the corresponding energy change ΔE
    - if $\Delta E > 0$: accept the spin-flip
    - if $\Delta E < 0$: accept the spin-flip with probability $\exp -\beta \Delta E$
- add the configuration to the Markov Chain

To start the algorithm, an initial configuration, which will generally not be in equilibrium, has to be chosen. It takes a while for the system to reach equilibrium, therefore the first configurations should be omitted. This chain of configurations approximates the canonical ensemble and can be used to study the properties of many-particle systems in equilibrium[8].[36][12]

The data set, used in this work, is an assembly of 20 datasets, which, through the application of the Metropolis Monte Carlo algorithm, are sampled from the configuration spaces accessible to the Ising model at 20 corresponding temperatures surrounding the critical temperature: $T/T_c \in [0.5, ..., 1.5]$, with lattice size $L = 16$ and periodic boundary conditions [36, 12, 24]. All training and testing sets that are used are subsets of this main data set which contains 10^4 configurations per temperature.

[8] In 1943, Lars Onsager succeeded in analytically solving the two-dimensional square-lattice Ising model for $H = 0$, using the transfer matrix method [6]. For $H \neq 0$ or higher dimensions, there is no analytical solution and the Ising model can only be studied numerically.

2 Principal Component Analysis

Principal Component Analysis is an unsupervised machine learning technique, that is used to find an optimal, linear, low-dimensional representation of a data sample.

For the square lattice Ising model, one data sample or configuration $\mathbf{x}$ can be represented as a point in N-dimensional space, since it consists of $N = L \times L$ individual spins.

$$\mathbf{x} = \begin{bmatrix} \sigma_1 & \sigma_2 & \dots & \sigma_{L\times L} \end{bmatrix} \tag{2.1}$$

We now wish to find $M < N$ new components, the principal components (PC_i). These are the M linear combinations weighting the original components $\sigma_i \in \mathbf{x}$, which summarize the configurations' most important information:

$$\mathbf{x}\mathbf{A}^* = \mathbf{y}^* = \begin{bmatrix} PC_1 & PC_2 & \dots & PC_M \end{bmatrix}, \tag{2.2}$$

with $\mathbf{A}^* \in \mathbb{R}^{N\times M}$. The columns $\mathbf{A}^*_{*i}$, with $i \in \{1, ..., M\}$, represent the weights of the principal components PC_i expressed in the original N-dimensional configuration basis. $\mathbf{y}^*$ is a low-dimensional representation of data sample $\mathbf{x}$, summarizing the total configuration of N spins with $M < N$ highly informative features.

2.1 Variance as a measure for importance

The question is, what determines an important feature? Without prior knowledge this is impossible to say. A data set of samples is required to form a reference. The data set represents the configuration space that one wishes investigate. To investigate the Ising model with zero magnetic-field for temperatures around the phase transition, the data set should be sampled from this configuration space (this data set was introduced in section 1.2.1).

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_S \end{bmatrix} = \begin{bmatrix} \sigma_{1,1} & \sigma_{1,2} & \dots & \sigma_{1,N} \\ \sigma_{1,1} & \sigma_{1,2} & \dots & \sigma_{1,N} \\ \vdots & \vdots & \ddots & \vdots \\ \sigma_{S,1} & \sigma_{S,2} & \dots & \sigma_{S,N} \end{bmatrix}, \tag{2.3}$$

represents a data set $\mathbf{X} \in \mathbb{R}^{S\times N}$ that contains S samples.

In this data set, some features will **vary** from sample to sample. Other features will be **constant** over the complete set (figure 2.1). The information that defines the individual samples lies in the variations between the different samples[1].

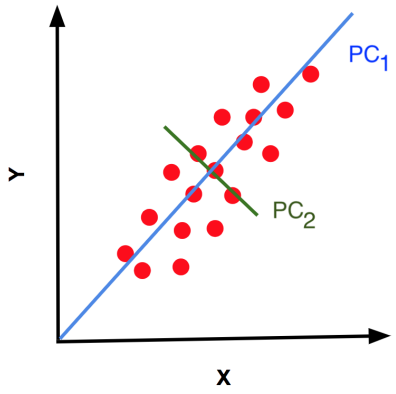


Fig. 2.1: Illustration of PCA. Firstly, the original two dimensional data set zero centered. Subsequently, components of the data are decoupled by performing an orthogonal transformation. The new axes lie along the direction for which the data varies the most (PC_1) and least (PC_2). Dimension reduction is done by omitting the least varying component. [28]

Therefore, for the M principal components, defined by the weights $\mathbf{A}_{*i}$ with $i \in \{1, ..., M\}$, which summarize the configurations most important information, it is required that:

1. $\text{var}_{\mathbf{x}\in\mathbf{X}}(PC_i = \mathbf{x}\mathbf{A}_{*i})$ is as high as possible.

2. Furthermore, features should not be correlated, since this implies that part of the information is stored multiple times, which is inefficient:
   $\text{covar}_{\mathbf{x}\in\mathbf{X}}(\mathbf{x}\mathbf{A}_{*i}, \mathbf{x}\mathbf{A}_{*j}) = 0$ for $i \neq j$.

However, some restrictions need to be added:

- The original components should not be rescaled, since this would correspondingly scale the variance and hence destroy the relation between importance and variance[2].

[1] An analogy to illustrate both the relation of variance with importance and the need for a well adapted data set: Suppose one wishes to investigate a certain bird. To determine the important features, a reference data set has to be defined. If one wants to find out what makes this bird unique in comparison with other birds, the data set will include all kinds of bird species. The constants; wings, a beak, feathers, are not important, every bird has them. The variances in size, color,... contain the real information. However, another objective, e.g. comparing the same bird to other animals, requires another, appropriate data set (containing all kinds of animals). Because the objective, and therefore also the data set, is different, very different conclusions about what is important will be drawn. Wings, beaks and feathers are no longer constants, they are unique to birds and therefore contain a lot of information.

[2] This restriction implies the assumption that the components of the original configuration basis are scaled in accordance to their importance. For the Ising model with zero magnetic field, this

- More strongly, to be able to use the concepts of variance and covariance in the new M-dimensional basis, the new components should always be introduced as a set of orthonormal linear combinations[3]. The orthogonal matrix $\mathbf{A} \in \mathbb{R}^{N\times N}$ (with $\mathbf{A}^T\mathbf{A} = \mathbf{1}$ or $\mathbf{A}^T = \mathbf{A}^{-1}$), is introduced as the expansion of $\mathbf{A}^* \in \mathbb{R}^{N\times M}$ to a complete orthonormal set. It can be proven that the orthogonal transformation of the data set preserves the total variance $\sum_{i=1}^{N} \text{var}_{\mathbf{x}\in\mathbf{X}}(\mathbf{x}\mathbf{A}_{*i})$.

Note that the restriction $\mathbf{A}^T\mathbf{A} = \mathbf{1}$, reduces the two requirements, listed above, to a single requirement. Since the decoupling of correlated features will always result in the increase of the variance of the feature that already varies most, at the cost of the other feature, which will decrease in variance (figure 2.1) [22, 15]. Therefore, finding the orthogonal transformation that decouples all features, automatically delivers the features with the highest and, for that reason, also the lowest variances.

In a preprocessing step of the data set $\mathbf{X} \in \mathbb{R}^{S\times N}$, the averages of every component $\langle\mathbf{x}\rangle_{\mathbf{x}\in\mathbf{X}} \in \mathbb{R}^{1\times N}$ are shifted to zero[4]. The covariance matrix of this zero-centered data set can easily be constructed as:

$$\mathbf{X}^T\mathbf{X} = \begin{bmatrix} \text{var}(\sigma_1) & \text{covar}(\sigma_1,\sigma_2) & \dots & \text{covar}(\sigma_1,\sigma_N) \\ \text{covar}(\sigma_2,\sigma_1) & \text{var}(\sigma_2) & \dots & \text{covar}(\sigma_2,\sigma_N) \\ \vdots & \vdots & \ddots & \vdots \\ \text{covar}(\sigma_N,\sigma_1) & \text{covar}(\sigma_N,\sigma_2) & \dots & \text{var}(\sigma_N) \end{bmatrix} \tag{2.4}$$

For the Ising model it was explained in section 1.1, that the hamiltonian introduces nearest neighbor interactions, which causes the spins to covary with their neighboring spins ($\text{covar}(\sigma_i,\sigma_j) \neq 0$) over a correlation length that depends on temperature. This spin correlation demonstrates that the representation of the configurations in the original spin basis, contains a lot of redundant information.

assumption is justified. All spins are equally important and they all have variance 1. However other data sets might require a preprocessing step to normalize their components, e.g. if they were measured at different scales.

[3] This restriction implies the assumption that the original configuration basis is orthonormal. For the Ising configurations, the basis set consists of the individual spins. It is meaningful to assume this an orthonormal basis because the physical correspondence between the two configurations can be measured their mutual distance as defined by this basis.

[4] Since the averages $\langle\mathbf{x}\rangle_{\mathbf{x}\in\mathbf{X}}$ are constants of the data set and the information lies in the variations on the data set, this zero-centering operation retains all information. For the Ising model, this preprocessing step is not necessary, because the average spins are already zero.

The key to finding the decoupled principal components, is to calculate the orthogonal matrix $\mathbf{A}$ which diagonalizes the covariance matrix $\mathbf{X}^T\mathbf{X}$ [5]:

$$\mathbf{Y}^T\mathbf{Y} = \mathbf{A}^T\mathbf{X}^T\mathbf{X}\mathbf{A} = \mathbf{\Lambda} = \begin{bmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \dots & \lambda_N \end{bmatrix} = \mathbf{A}^{-1}(\mathbf{X}^T\mathbf{X})\mathbf{A}, \tag{2.5}$$

where the variances $\lambda_i = \text{var}(PC_i)$ are sorted in descending order. As a result, the columns of $\mathbf{A} \in \mathbb{R}^{N\times N}$ will contain the coefficients of the principal components in descending order of importance. The dimensionality reduction matrix $\mathbf{A}^* \in \mathbb{R}^{N\times M}$, simply contains the first M columns of $\mathbf{A}$.

2.2 Results

The analysis is done on two training sets, $\mathbf{X}_{train,100}$ and $\mathbf{X}_{train,400}$, which contain respectively 100 and 400 Ising configurations per temperature, delivering two distinct orthogonal transformations $\mathbf{A}_{100}$ and $\mathbf{A}_{400}$. The performance of both transformations was tested on $\mathbf{X}_{test}$ which contains 400 configurations per temperature. For a well performing transformation $\mathbf{A}$, the covariance matrix of the transformed test set should approach the diagonalized matrix (equation 2.5), $\mathbf{\Lambda}_{train}$, as well as possible:

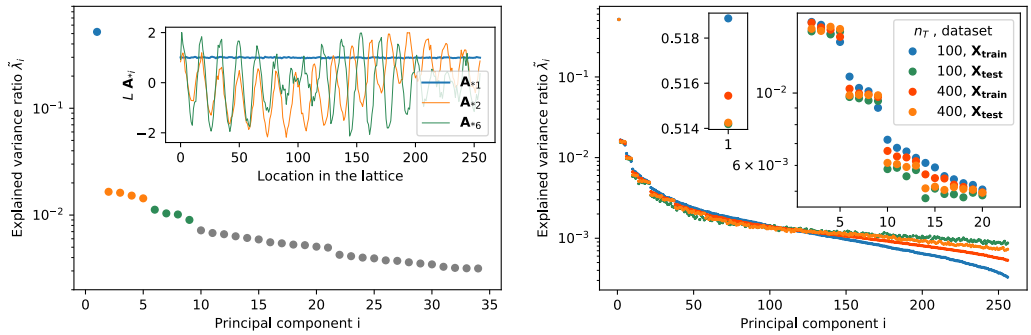
$$\mathbf{\Lambda}_{test} = \mathbf{A}^T\mathbf{X}_{test}^T\mathbf{X}_{test}\mathbf{A} \approx \mathbf{A}^T\mathbf{X}_{train}^T\mathbf{X}_{train}\mathbf{A} = \mathbf{\Lambda}_{train}. \tag{2.6}$$

Figure 2.2a shows the explained variance ratios

$$\tilde{\lambda_i} = \frac{\lambda_i}{\sum_{j=1}^{N}\lambda_j} \tag{2.7}$$

of the first 35 principal components, calculated on $\mathbf{X}_{train,100}$. In figure 2.2b, all L^2 explained variance ratios are shown for the orthogonal transformations $\mathbf{A}_{100}$ and $\mathbf{A}_{400}$ on both the training and the testing configurations. Prominently, there is only one dominant principal component, which already explains half of the variation (51%). All other principal components have far lower variations ($< 2\%$), indicating that a nonlinear function is probably more adequate to describe the second half of the variance.

[5] An easy way to perform a diagonalization for a matrix of the form $\mathbf{X}^T\mathbf{X}$, is via a singular value decomposition of $\mathbf{X}$.



(a) (b)

Fig. 2.2: (a) The explained variance ratios $\tilde{\lambda}_i$ for the first 35 principal components for a training set containing 100 configurations per temperature. In the inset, the weight vectors, $\mathbf{A}_{*1}, \mathbf{A}_{*2}$ and $\mathbf{A}_{*6}$ are displayed, indicating how the spins in the lattice are weighted to find the corresponding principal components PC_1, PC_2 and PC_6. (b) The explained variance ratios $\tilde{\lambda}_i$ for all L^2 principal components are shown for two training sets containing 100 and 400 configurations per temperature (blue and red). The resulting orthogonal transformations $\mathbf{A}_{100}$ and $\mathbf{A}_{400}$ are subsequently used to evaluate the testing set (green and orange). The linear lattice size $L = 16$.

In the inset of figure 2.2a, the weight vector, $\mathbf{A}_{*1}$, of the first principal component is plotted, showing how the individual spins are weighted to calculate the first principal component, PC_1. All spins have an equal contribution, hence PC_1 is proportional to the magnetization of the lattice:

$$PC_1 = \mathbf{x}\mathbf{A}_{*1} = \frac{1}{\sqrt{N}} \sum_{i=1}^{N} \sigma_i \sim m = \frac{1}{N} \sum_{i=1}^{N} \sigma_i \tag{2.8}$$

It is no coincidence that the magnetization establishes the most varying linear combination of spins, since it is the order parameter of the Ising model, changing rapidly around the critical temperature T_c from approximately 1 in the ferromagnetic phase to approximately[6] 0 in the paramagnetic phase (figure 2.3a).

Even though it is established that there is only one dominant principal component, corresponding to the magnetization m, it is still worthwhile to take a look at the other components. In figure 2.2, the explained variances seem to be ordered in groups, hinting that $(PC_2 - PC_5)$ and $(PC_6 - PC_9)$ share their level of importance. Taking into account the periodic boundary conditions on the configurations, it was found that the weight vectors corresponding to one group are mere translations of one another, proving their physical equivalence (figure 2.4).

In reference [15], it was established that the weight vectors $\mathbf{A}_{*i}$, which define the principal components, correspond to the Fourier modes of the spin configuration. This result is reflected in the weights displayed in figure 2.4. The transformation

[6] This is for finite lattices, for infinite lattices, m is exactly 0 in the paramagnetic phase.

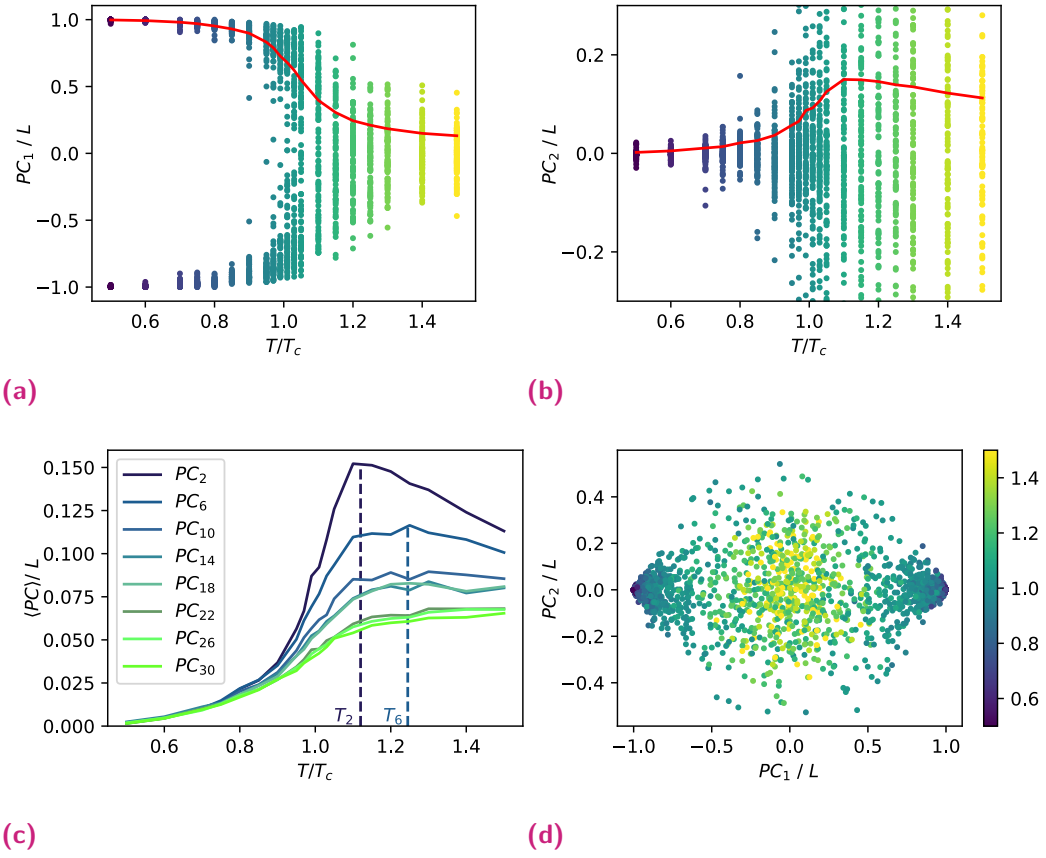


Fig. 2.3: In (a) and (b), PC_1 and PC_2 are respectively displayed as a function of temperature for all configurations in the testing set. In red, their average absolute value $\langle |PC_i| \rangle$ is shown. In (c) the average absolute values of the higher order principal components is displayed as a function of temperature. (d) The projection of N-dimensional spin configurations onto the two-dimensional subspace, spanned by the first two principal components.

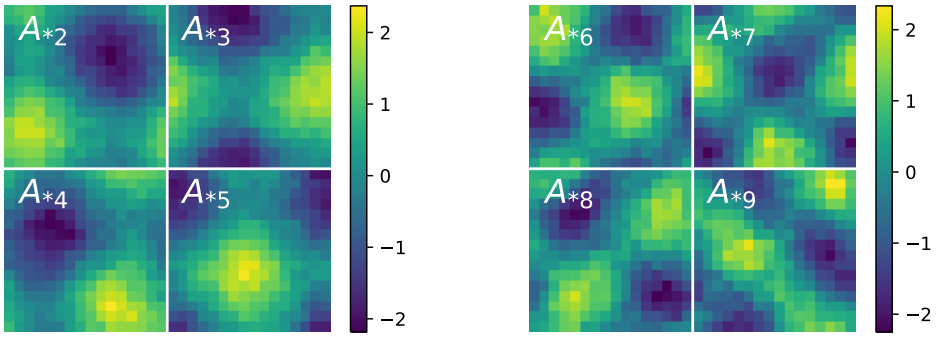


Fig. 2.4: The weight vectors are ordered in groups that display translational invariance and are therefore physically equivalent. (a) $\mathbf{A}_{*2} - \mathbf{A}_{*5}$ (b) $\mathbf{A}_{*6} - \mathbf{A}_{*10}$.

$\mathbf{xA} = \mathbf{y}$, can therefore be understood as the fourier decomposition of spin configuration $\mathbf{x}$ In the ordered phase $(T < T_c)$, the decomposition is dominated by the first principal component: a constant value over the entire configuration. Close to transition, $(T = T_c)$, the other principal components, arranged according to decreasing order (increasing frequency f), gain importance (figure 2.3c). These principal components can be seen as disorder indicating parameters, however their variation is less strong then the variation of PC_1. The average absolute values of the principal components, $\langle |PC_i| \rangle$, reach a maximum at a specific temperature T_i in the paramagnetic phase, T_i could be interpreted as the temperature for which the correlation length $\frac{1}{f_i}$ is dominant.

Figure 2.3d displays the projection of N-dimensional spin configurations onto the two-dimensional subspace, spanned by the first two principal components. Most variation is oriented along the first principal axis reflecting once again the dominance of PC_1. The configurations are ordered in three groups: a high temperature cluster in the center, corresponding to the paramagnetic phase and two low temperature clusters at $|PC_1| = 1$, corresponding to the ferromagnetic phase. We can conclude that, through the deduction of the order parameter m and, with limited influence, the deduction of the other, disorder indicating principal components, PCA is able to locate the phase transition and order the Ising configurations in their respective phase.

For a small range of temperatures around T_C, many configurations are difficult to classify in either one of the clusters. This can be ascribed to the large variations of the magnetization per spin m around T_c for finite lattice sizes. For increasing lattice size L, the region of confusion grows smaller and this leads to more distinguishable clusters [15, 40][7].

Even though the interpretation of the higher order principal components increases insight, it should be remembered that only PC_1 explains a significant share of the total variance. This implies that, apart from the magnetization, the highly varying features cannot be written as a linear combination of the spins. A first improvement could entail the merging of the equivalent principal components. The combination of principal components $PC_2 - PC_5$ as

$$\sqrt{(PC_2)^2 + (PC_3)^2 + (PC_4)^2 + (PC_5)^2}, \tag{2.9}$$

represents 6% of the total variance in one non linear component instead of four linear components. For further improvements, more subtle non linear functions of the spin configuration will be needed, this requirement is incompatible with the

[7] This phenomenon can be used to perform a finite size scaling analysis in order to find the critical components of the infinite system [15, 40].

linear technique PCA. As an illustration, the total energy function from which a lot of information on the phase transition can be inferred, could never be found by linear techniques such as PCA, since it contains of a non linear spin-spin interaction term. Other, non linear unsupervised learning techniques are more adequate to perform this task. In reference [7], stochastic neighbor embedding (t-SNE) was used to present a non linear two-dimensional visualization of the Ising configurations.

From figure 2.2b, the influence of the training set size can be inferred. In blue and green, the performance of $\mathbf{A}_{100}$ is displayed. The inset shows that the explained variance ratios of the first principle component, $\tilde{\lambda}_1$, is higher for $\mathbf{X}_{train,100}$ then for the $\mathbf{X}_{test}$. The discrepancy between the train and test results indicate that the transformation $\mathbf{A}_{100}$ is overfitted. For $\mathbf{A}_{400}$ (red and orange), the overfitting is strongly reduced. Related, the differences on the explained variance ratios of the groups of equivalent components, is most visible for $\mathbf{X}_{test}$ followed by $\mathbf{X}_{train,400}$ and least visible for $\mathbf{X}_{train,100}$. The reason is that larger training sets lie closer to the real configuration space, which makes the model less receptive to random noise on the training set.

2.3 Generalization of terminology

In this section, it will be demonstrated that Principal Component Analysis is perfectly compatible with the description of Machine Learning, given in chapter 1.1 [38]:

A **model** $f(\mathbf{x}|\boldsymbol{\theta})$ is required to perform a certain **task** on configuration $\mathbf{x}$ with respect to the **data set** $\mathbf{X}$. To measure the models degree of performance, a **cost function** $C(f(\mathbf{X}|\boldsymbol{\theta}))$ is introduced. The learning consists of finding the optimal parameters $\boldsymbol{\theta}^*$ for which the cost is minimal:

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}}\{C(f(\mathbf{X}|\boldsymbol{\theta}))\}. \tag{2.10}$$

To perform the minimization, a **minimization method** is needed.

The task

The task of PCA is to find a low-dimensional representation $\mathbf{y}^* \in \mathbb{R}^M$ of a data sample $\mathbf{x} \in \mathbb{R}^N$.

The model

The model $f(\mathbf{x}|\boldsymbol{\theta})$, corresponds to an orthogonal transformation $\mathbf{A} \in \mathbb{R}^{N\times N}$, followed by the selection of the first M components to reduce the dimension:

$$f : \mathbb{R}^N \to \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{y}^* = f(\mathbf{x}|\boldsymbol{\theta}) = (f_2 \circ f_1)(\mathbf{x}|\boldsymbol{\theta}) \tag{2.11}$$

$$f_1 : \mathbb{R}^N \to \mathbb{R}^N, \mathbf{x} \mapsto \mathbf{y} = f_1(\mathbf{x}|\boldsymbol{\theta}) = \mathbf{x}\,\mathbf{A} \tag{2.12}$$

$$f_2 : \mathbb{R}^N \to \mathbb{R}^M, \mathbf{y} \mapsto \mathbf{y}^* = [y_1, y_2, \ldots, y_M]. \tag{2.13}$$

The parameters $\boldsymbol{\theta}$ of the model can be brought into correspondence with the $\frac{N(N-1)}{2}$ degrees of freedom for an orthogonal matrix.

The data set

The performance of the model can only be defined with respect to the configuration space of interest. Therefore a data set $\mathbf{X}$, that approaches this configuration space as well as possible, is required.

The cost function

As explained in section 2.1, the variance of a feature $y_i = \mathbf{x}\mathbf{A}_{*i}$ is a measure for the features importance.

Remark that in reality, diagonalization of the covariance matrix intrinsically ensures optimal performance on the training set, whitout the need for a cost function. However, it is still interesting to construct a cost function to see the analogy with other machine learning techniques [38].

The cost-function could be defined as:

$$C(f(\mathbf{X}|\boldsymbol{\theta})) = 1 - \frac{\sum_i^M \mathrm{var}_{\mathbf{X}}(y_i)}{\sum_i^N \mathrm{var}_{\mathbf{X}}(y_i)} = 1 - \frac{\sum_i^M (\mathbf{Y}^T\mathbf{Y})_{ii}}{\mathrm{tr}(\mathbf{Y}^T\mathbf{Y})} = 1 - \frac{\sum_i^M (\mathbf{A}^T\mathbf{X}^T\mathbf{X}\mathbf{A})_{ii}}{\mathrm{tr}(\mathbf{A}^T\mathbf{X}^T\mathbf{X}\mathbf{A})} \tag{2.14}$$

However, this costfunction will solely divide the configuration-space in an M-dimensional maximally varying subspace and an uncorrelated $(N-M)$-dimensional

minimally varying subspace. The covariance matrices of the subspaces will not be diagonalized, nor will the components be ordered according to decreasing variance.

A possibility for a cost-function that does diagnonalize the complete covariance matrix, is:

$$C(f(\mathbf{X}|\boldsymbol{\theta})) = \sum_{i}^{N} \sigma_{\mathbf{X}}(y_i), \tag{2.15}$$

For every orthogonal transformation $\mathbf{A}$, the total variance: $\text{tr}(\mathbf{A}^T\mathbf{X}^T\mathbf{X}\mathbf{A}) = \sum_i^N \text{var}_{\mathbf{X}}(y_i)$ will be the same constant. Therefore, the sum of the square roots $\sum_i^N \sigma_{\mathbf{X}}(y_i)$ will be minimal when the differences between the individual components $\text{var}(y_i)$ are maximalized. This cost-function might however also not be the good choice, because the minimum is rather shallow. Furthermore, the components of the resulting matrix $\mathbf{A}$ should still be sorted in decreasing order of variance.

Another approach could be to resolve the M most varying features subsequently. One starts by extracting the most varying component from the original configuration space with a cost function of type (2.14). Once this component is found, it is substracted of the original configuration space which is then reduced to dimension $N-1$. From this subspace, the second most varying component is extracted and so on.

The minimization method

The used minimization method is a analytical single step diagonalization through singular value decomposition. However, this analytical approach becomes very slow for increasing data set sizes, therefore, for large data sets, iterative methods are preferred. [30]

3 Neural Networks

3.1 Introduction

In the first sections of this chapter neural networks (NN) are introduced as candidates to represent a model that can be trained through machine learning[22][3]. Apart from the model type, other key components to determine the learning process (as defined in the introduction on machine learning) are the cost function, as a measure of the model's performance and the minimization method to minimize this cost. These components are treated in the final sections of this chapter. Especially the selection of the cost function is very much related to the type of task you want the model to perform: classification, regression, dimensionality reduction, etc.

3.2 Building block: the neuron

A neural network (NN) is a computing system that processes complex data inputs and returns an output. As the name suggests, neural networks are loosely inspired by the biological neural networks that constitute our brains [25]. They lend themselves to be graphically represented as a network, built up out of nodes and links (figure 3.1). To *read* this graphical representation, let's start with the basic building block of

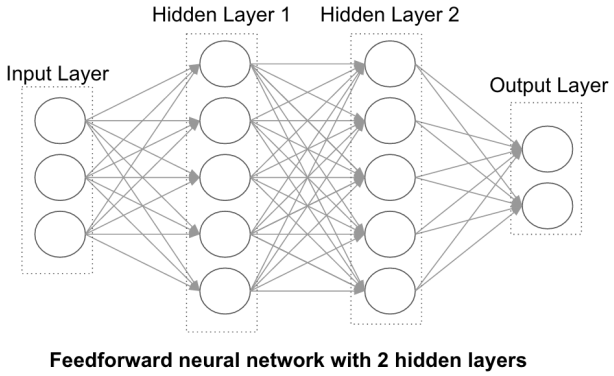


Fig. 3.1: The architecture of a fully connected neural network, with one input layer, one output layer and two hidden layers. (source figure: [1])

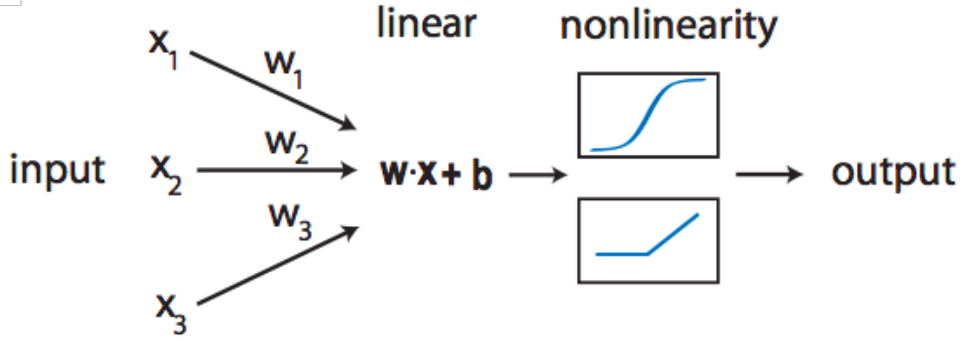


Fig. 3.2: Representation of a neuron, the building block of neural networks. The neuron is composed out of two parts. First, a linear transformation calculates a weighted sum ($\mathbf{w}\,\mathbf{x} + b$) of the input-vector $\mathbf{x}$. Secondly, a nonlinear activation function is applied. (source figure [22])

a NN: a neuron (figure 3.2).

A neuron $\phi(\mathbf{x}|\boldsymbol{\theta})$ is the most basic neural network. The neuron is graphically represented (figure 3.2) by a node with N incoming links and one outgoing link, processing an input-vector $\mathbf{x}$ of length N and generating a corresponding scalar output y. Every neuron ϕ[1] is a composition of two functions: a linear transformation T and a nonlinear activation function σ

$$\phi : \mathbb{R}^N \to \mathbb{R}, \mathbf{x} \mapsto y = \phi(\mathbf{x}|\boldsymbol{\theta}) = (\sigma \circ T)(\mathbf{x}|\boldsymbol{\theta}). \tag{3.1}$$

Firstly, the linear transformation T is applied to the vector $\mathbf{x}$:

$$T : \mathbb{R}^N \to \mathbb{R}, \mathbf{x} \mapsto z = T(\mathbf{x}|\boldsymbol{\theta}) = T(\mathbf{x}|\mathbf{w}, b) = \mathbf{w}\,\mathbf{x} + b, \tag{3.2}$$

where parameters $\boldsymbol{\theta}$ correspond with a weight-vector $\mathbf{w}$ and a scalar bias b. Through multiplication with the weight-vector, $\mathbf{w}$, every input component x_i is evaluated with relative importance w_i. Furthermore, a bias b is included in the linear transformation. The scalar output z is called the *weighted sum*. Graphically, every incoming link can be ascribed a corresponding weight w_i, for $i \in \{1, .., N\}$.

Secondly, a nonlinear activation function σ is applied to the weighted sum $\mathbf{w}\,\mathbf{x} + b$, returning a scalar output $y = \sigma(\mathbf{w}\,\mathbf{x} + b)$. This output is called the *activation* of the neuron. The activation function is crucial to introduce non-linearity in the neuron

[1] a neuron $\phi(\mathbf{x}|\boldsymbol{\theta})$, and every model in general, is trained to be optimized w.r.t. its parameters $\boldsymbol{\theta}$. It is important to note however, that these parameters $\boldsymbol{\theta}$ are part of the model and determine the models performance. So, when a model processes a certain input $\mathbf{x}$, the parameters $\boldsymbol{\theta}$ are treated as constants.

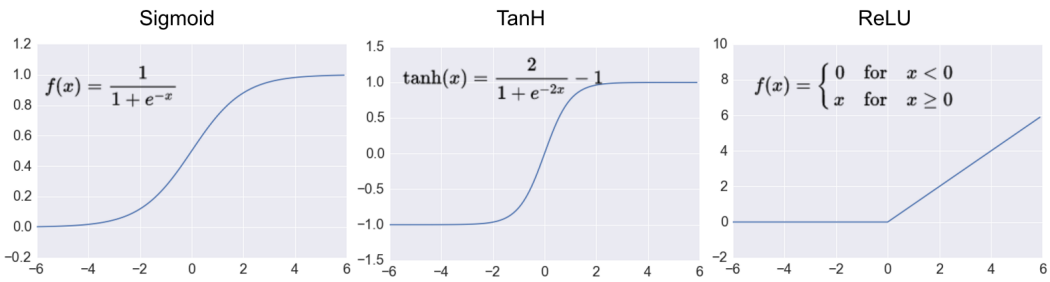


Fig. 3.3: This figure shows three important activation functions: sigmoid, hyperbolic tangent (Tanh) and rectified linear unit (ReLU). (source figure: [1])

and hence in the total neural network. (elaborated in section 3.3.2)

Figure 3.3 depicts the most common activation functions: Sigmoid, Tanh and ReLU. Different choices of activation functions lead to different neuron-properties. The earliest neural networks used the Heaviside-function as activation. But, these neural networks were incompatible with the powerful gradient descent minimization method, because the Heaviside is undifferentiable [25]. Smoothed versions of the Heaviside, Tanh and Sigmoid, became the norm. However, when the input parameters become large, the gradients of Tanh and Sigmoid approach zero, therefore training through gradient descent can still be difficult. Today, ReLU is a more common activation function in the hidden layers. In contrast to Sigmoid an Tanh, the ReLU activation-function, has a finite gradient even for large input parameters [22]. For binary classification purposes, Sigmoid is often used in the output-layer, to ensure that the output represents a probability (section 3.7.1).

3.3 Universal Approximation Theorem

Let's turn our attention back to the full neural network (figure 3.1). This network consists of multiple nodes, ordered in layers. The first and last layer are identified as the input- and output- layer, all layers in between are referred to as the hidden layers.

With the exception of the input nodes, each node, together with its incoming links, forms a neuron. Note that there are no cycles in the network [2] , information travels in one direction only, the output of one layer serves as input for the next.

[2] In fact, the neural network discussed here, is a feed-forward neural network. There are other types of NNs in which such cycles do appear, namely recurrent neural networks (RNN). However, these *cycles* do not represent a real closed loop, since they are directed in time[4].

The neural network $f(\mathbf{x}|\boldsymbol{\theta})$ corresponds with the function

$$f : \mathbb{R}^N \to \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{y} = f(\mathbf{x}|\boldsymbol{\theta}), \tag{3.3}$$

where $\mathbf{x}$ is an N-dimensional input-configuration and $\mathbf{y}$ is the M-dimensional corresponding output.

It is an interesting question to ask ourselves what classes of functions can be represented by these feed-forward neural networks.

Before jumping to conclusions in the form of the universal approximation theorem (UAT), let's do two thought-experiments. In the first experiment, the representational power of a NN without hidden layers is investigated. Secondly, a NN without activation functions is considered.

3.3.1 Network without hidden layers

Consider a network without hidden layers, consisting solely of an N-dimensional input and an M-dimensional output layer. The network will have the following form:

$$f : \mathbb{R}^N \to \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{y} = f(\mathbf{x}|\mathbf{W}, \mathbf{b}) = \sigma(\mathbf{W}\,\mathbf{x} + \mathbf{b}), \tag{3.4}$$

where the parameters $\mathbf{w}_i \in \mathbb{R}^N$and $b_i \in \mathbb{R}$, with $i \in \{1, .., M\}$, that make up the M neurons determining the output layer, are aggregated in $\mathbf{W} \in \mathbb{R}^{N \times M}$ and $\mathbf{b} \in \mathbb{R}^M$.

Note that equation (3.4) is in fact a general representation for the working of a single NN-layer[3]. Consistent with the terminology used for neurons, $\mathbf{z} = \mathbf{W}\,\mathbf{x} + \mathbf{b}$ and $\mathbf{y} = \sigma(\mathbf{z})$ are respectively the weighted sum and the activation of this layer.

This function can describe any linear transformation $\mathbf{W}$ accompanied by a certain translation $\mathbf{b}$ and nonlinear evaluation of the output. To illustrate the two-layered networks limited representational power, consider a simple property of the form $(\mathbf{w}_1\mathbf{x})^2 + (\mathbf{w}_2\mathbf{x})^2$; it can not be reduced to a linear transformation evaluated by a non-linear function and therefore it can never be described by this two-layer network.

[3]This **excludes** the input layer, which solely serves to pass on the input. This layer is not associated with any parameters, therefore it can not be expressed in the form of (3.4).

3.3.2 Network without activation layers

Consider a network, without activation functions and three layers: an N-dimensional input layer, a P-dimensional hidden layer and an M-dimensional output layer. The networks corresponding function will then have the following form:

$$f : \mathbb{R}^N \to \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{y} = f(\mathbf{x}|\boldsymbol{\theta}) = (f_2 \circ f_1)(\mathbf{x}|\boldsymbol{\theta}). \tag{3.5}$$

f_1 represents the working of the hidden layer which processes the input and f_2 represents the working of the output layer, processing the activation of the hidden layer:

$$f_1 : \mathbb{R}^N \to \mathbb{R}^P, \mathbf{x} \mapsto \mathbf{y} = f_1(\mathbf{x}|\mathbf{W}_1, \mathbf{b}_1) = \mathbf{W}_1\mathbf{x} + \mathbf{b}_1 \tag{3.6}$$

$$f_2 : \mathbb{R}^P \to \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{y} = f_2(\mathbf{x}|\mathbf{W}_2, \mathbf{b}_2) = \mathbf{W}_2\mathbf{x} + \mathbf{b}_2 \tag{3.7}$$

where $\mathbf{W}_1$, $\mathbf{b}_1$ and $\mathbf{W}_2$, $\mathbf{b}_2$ determine the weights, biases of the hidden and output layer respectively. Evaluating the total function

$$f : \mathbb{R}^N \to \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{y} = \mathbf{W}_2(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1) + \mathbf{b}_2 = (\mathbf{W}_2\mathbf{W}_1)\mathbf{x} + (\mathbf{W}_2\mathbf{b}_1 + \mathbf{b}_2) \tag{3.8}$$

indicates that without activation functions, every neural network can be reduced to a simple two-layer neural network describing a linear transformation $\mathbf{W}$ accompanied by a certain translation $\mathbf{b}$. This means that activation functions are absolutely necessary if we wish to introduce nonlinearity in the model.

3.3.3 Universal Approximation Theorem

General feed-forward neural networks with both hidden layers and activation functions are more complicated. They have been investigated thoroughly in the past[14][38]. These studies led to very interesting results that were concisely summarized in the Universal Approximation Theorem (UAT):

Feed-forward networks, with one or more hidden layers containing sufficiently many* neurons, can approximate any continuous function on compact[4] subsets of $\mathbb{R}^N$ Furthermore, this is possible under very general conditions* on the activation function.*

Some additional clarifications are in order:

- one or more, sufficiently many: The theorem does not say anything on the optimal number of hidden layers and neurons per layer.

[4] closed and bounded

- very general conditions on the activation function: It should be emphasized that this does not mean that all activation functions will perform equally well in specific learning problems (section 3.2).

To conclude: neural networks have an enormous representational power. They can approximate every conceivable physical function, making neural networks universal and characteristic machine learning models. The generality of neural networks allows to investigate numerous different and complex problems with the same tools.

It should be noted however, that this extensive representational capacity does not always make neural networks the best-adapted model-choice. Especially for tasks that can be fulfilled with a simpler model (e.g. approximating a linear phenomenon), it is advantageous to use this simpler model, which is more adapted to the specific task and will be trained more easily. This will be elaborated further in section 3.5.

3.4 Fully Connected Neural Networks

The neural networks that were discussed until now, are in fact fully connected neural networks.[5] A neural network is termed fully connected when

- every neuron in the network is connected through a weighted link to all the neurons from the previous layer

- the weights and biases of every neuron are independent parameters.

To develop neural networks, a machine learning library for python, PyTorch, was used. One of the models that has been constructed is the fully connected neural network represented in figure 3.4. The model is intended to be trained for classification of Ising configurations (drawn from the data set that was presented in the introductory chapter). The returned value y represents the models estimated probability for the configuration to be in the ferromagnetic phase, $1 - y$ represents the probability of being in the paramagnetic phase.

When examining the topology of a fully connected neural network, it becomes clear that the multi-dimensional structure of the input configurations is not used. This is why, in programming, all input-configurations are represented by a one-dimensional

[5] These fully connected networks are equally referred to as multilayer perceptrons MLP or 'vanilla' neural networks. [13][26]

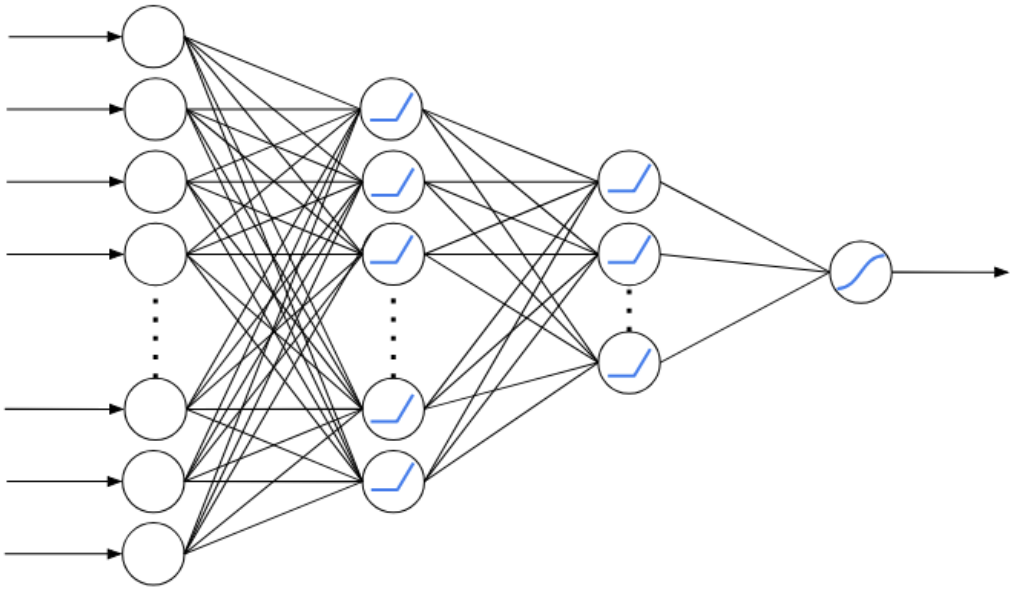


Fig. 3.4: Architecture of a fully connected model with an input layer, 2 hidden layers and an output layer, containing respectively 256, 100, 10 and 1 nodes. The hidden layers have ReLU activation functions and the output-neuron has a sigmoid activation function (to ensure that the the scalar output y can represent a probability $\in [0, 1]$.)

vector. To illustrate; the $16 \times 16-$shaped Ising configurations are flattened out to a one-dimensional input-vector of length 256. The model should learn that this structure is present through training, but this knowledge is not intrinsically incorporated in its architecture.

3.5 Steering the model

When selecting a model that will be trained to perform a certain task, it might be worthwhile to consider any additional information (e.g. periodicity, symmetry, translation invariance of the input-configuration or the hamiltonian, etc.) that can be implemented as general constraints[6] on the model, to steer it in the right direction. This approach can have several advantages:

- The model's architecture is altered based on knowledge and insight in the problem. Therefore, it will recognize physical trends more easily and is less sensitive to noise on the training data. This makes the model more performant and a better generalizer.

[6]Here we run into a tension field. What should be understood by *general constraints*? How much should the model be steered? What is to gain? A better and potentially faster performing model, increasing insight in the problem. What is to lose? Time spent programming and a general, non-biased model that was not influenced by possibly false assumptions of the programmer on poorly understood, chaotic data.

- Strongly related, the model requires less parameters for the same task, because part of the information is already incorporated in the models structure. In general, it takes less data to train a model with less parameters.

- Thoughtful adaptation of the model, increases insight in its workings. (A nice example is the visualization of kernels, or 'feature-detectors' in convolutional neural networks, section 4.1.4)

Let's illustrate this point with a simple example. Suppose a model has to be trained to approximate a certain phenomenon, for which you know from physical considerations that it is linear. Despite the universal approximation theorem, which assures that a general neural network can approximate this function, it is obvious that a linear transformation will be the better choice for the model because it incorporates the physics (in this case the linearity) directly into the models architecture.

Another example is treated in the next section: Convolutional neural networks (CNN). Compared to fully connected neural networks, CNN intrinsically incorporate additional information into their architecture.

3.6 Convolutional Neural Networks

A convolutional neural network (CNN) is a class of feed forward neural networks. Convolutional neural networks differ from fully connected neural networks because they intrinsically encode specific spatial information of their input-configuration[7] into their architecture. Most links are omitted and groups of neurons can share weights and biases. This significantly reduces the number of learnable parameters.

To illustrate what is meant by spatial structure , think of the finite, square lattice Ising configurations. These configurations are 2-dimensional. Furthermore, by closely examining the expression for the energy for the Ising model with zero external magnetic field (chapter 1.2.1)

$$E = -J \sum_{i,j=nn(i)}^{N} s_i s_j, \tag{3.9}$$

we find:

[7] The input-configuration is usually a 2-dimensional image or configuration, but 1- and 3-dimensional inputs also occur.

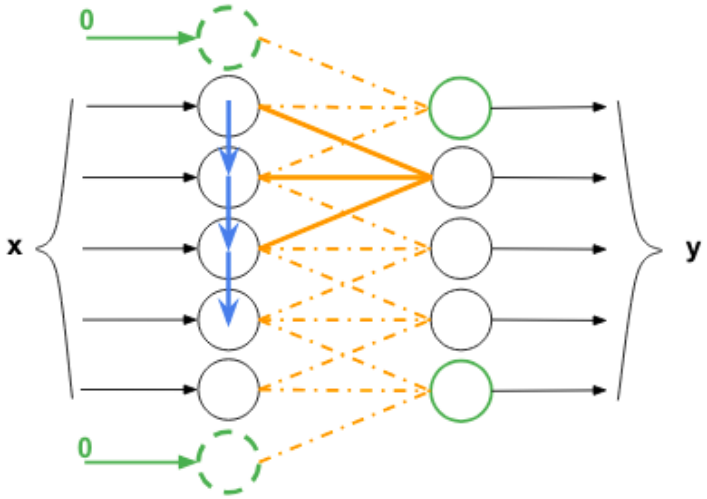


Fig. 3.5: Representation of a one dimensional convolution layer. The output is calculated as the convolution of the input with a kernel of size 3. Orange: kernel with kernelsize K = 3. Blue: stride S = 1. Green: zero padding with P = 2 to maintain the spatial structure of the input-configuration.

1. Locality: The phase transition is made possible by local interactions. The spins are locally correlated and these correlations contain important information on the thermodynamic properties of the system.

2. Homogeneity: Every spin is evaluated equally, independent of its location on the grid.

Neither of these properties is accounted for in the fully connected model. The convolutional neural network introduces both properties into the architecture of the network.

Apart from the use in statistical models such as the Ising model, CNN is a very important tool in computer vision. Computer vision studies ML-models with the task of analyzing images. A machine can recognize a certain object by identifying it with a collection of features that are linked to this object (e.g. the features eyes, nose and mouth help to recognize a face). These features occur locally, encoded in the correlations between neighboring pixels. Furthermore, it doesn't matter where the features occur (homogeneity). E.g. an eye might occur in the middle or in the corner of a picture, but in both cases, it should be recognized as an eye[8].[22][9]

Figures 3.5 and 3.6 show a few of simple convolution layers in 1 and 2 dimensions respectively. A convolution layer $f_{conv}(\mathbf{x}|\mathbf{w}, b)$ calculates the convolution of the

[8]This assumption does not always make sense. E.g. when the input solely contains images in which the faces have been centered, the edge and center of the image should be treated in different ways.

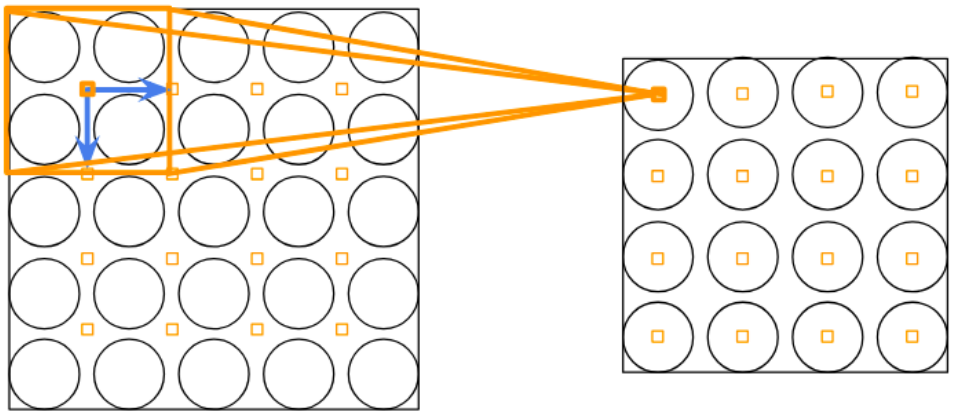


Fig. 3.6: Representation of a two dimensional convolution layer. Orange: kernel with kernelsize K $= 2 \times 2 \triangleq 2$, Blue: stride S $= 1 \times 1 \triangleq 1$

input **x** with a certain function, termed a kernel. The kernel acts as a *local* filter: it takes as inputs a *small spatial patch* of the total input and multiplies these inputs with its own fixed set of weights **w**. The kernel strides over the complete input-plane. With every stride, it performs the *same analysis at a different location*. The complete layer is complemented with an activation function σ. In the 1-dimensional case, this results in a function of the following form:

$$\mathbf{f}_{conv} : \mathbb{R}^N \to \mathbb{R}^M, \mathbf{x} \mapsto \mathbf{y} = \sigma(\mathbf{w} *_S \mathbf{x} + b\ \mathbf{1}), \tag{3.10}$$

where '$*_S$' represents a convolution with stride S, which can be written out in full:

$$y_i = f_{conv,i} = \sigma(\mathbf{w}\ \mathbf{x}[iS : iS + K] + b), \tag{3.11}$$

with kernel size K: $\mathbf{w} \in \mathbb{R}^K$, $i \in \{0, .., M-1\}$ and $N = S(M-1) + K$. This expression can easily be generalized for higher dimensions.

Roughly stated, one kernel learns to detect one feature of the input. For images, the kernels could be edge or color detectors. For Ising-configurations, the kernels are more difficult to interpret but they can roughly be divided into order and disorder detectors (section 4.1.4). A convolution layer usually evaluates its input with more than one kernel. The use of multiple kernels gives rise to an extra dimension per layer, the depth D, to store multiple channels (figure 3.7). The total shape of a layer with spatial size $N \times N$ will be written down as $D@N \times N$. These channels are *'fully connected'*, meaning that the transition from a layer with depth D_1 to a layer with depth D_2 requires $D_1 \times D_2$ unique kernels. A two-dimensional, convolutional layer

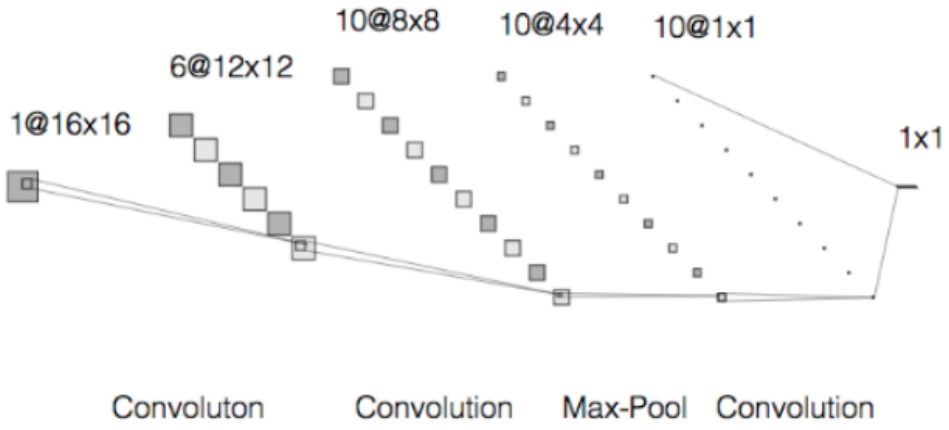


Fig. 3.7: A convolutional network, consisting of respectively two convolution layers, a pooling layer, another convolution layer and a fully connected output-layer. The layers are ordered in three dimensions $D@N \times N$, where D indicates the number of channels and $N \times N$ represent the spatial dimensions per channel. As a reference: figure 3.6 represents one channel to channel transition between successive layers.

with multiple channels can be written as follows (generalization of equation (3.10)):

$$\mathbf{f}^{l}_{conv} : \mathbb{R}^{D_{l-1} \times N_{l-1} \times N_{l-1}} \rightarrow \mathbb{R}^{D_l \times N_l \times N_l}, \mathbf{Y}^{l-1} \mapsto \mathbf{Y}^{l}, \tag{3.12}$$

where the activation per channel is determined by:

$$\mathbf{Y}^{l,j} = \sigma(\sum_{k=1}^{D_{l-1}} \mathbf{W}^{l,j,k} *_S \mathbf{Y}^{l-1,k} + b^j \ \mathbf{1}), \tag{3.13}$$

with $j \in \{1, .., D_l\}$. $\mathbf{Y}^{l,j} \in \mathbb{R}^{N_l \times N_l}$ represents the activation of layer l channel j and $\mathbf{W}^{l,j,k} \in \mathbb{R}^{K \times K}$ represents the kernel, with kernel size K, from channel k in layer $l-1$ to channel j in layer l.

A CNN is build out of consecutive convolution layers accompanied by a method to reduce the spatial size. Analysis of CNNs suggests that the first layers of the neural networks learn elementary features on a small, local scale. The subsequent layers then combine these elementary features into more abstract, complicated features on a larger scale. E.g. in computer vision, the recognition of a face might proceed as follows: The first layer detects edges, dots, stripes, etc. The second layer combines these elementary features into eyes, a nose and a mouth. The final layer recognizes this collection as a face. [22]

3.6.1 Reducing spatial size

To reduce the spatial size of the layers in the CNN, it is common to either insert a **pooling** layer in between successive convolution layers or to use a convolution layer with a **stride S** $\geq$ **2**.

A pooling layer operates independently on every channel. The spatial size is reduced by applying a filter with a certain stride. This filter selects a local patch of the total input and applies a pooling function, e.g. taking the maximum, averaging, etc. Max pooling is most commonly used based on experimental results.[11]Note that the pooling layer contains no learnable parameters.

The use of a convolution layer, with stride S $\geq$ 2, has the advantage that the size-reduction is learnable, the kernels can be viewed as trained pooling filters.

3.6.2 Padding

Equation (3.11) illustrates that the input and outputs spatial size N and M are connected through the following expression:

$$N = S(M-1) + K. \tag{3.14}$$

This implies restrictions on the possible choices for S, K and the spatial size of the output M. To increase control over the spatial output size, one can add an extra border or padding with size P to the input-configuration:

$$N = S(M-1) + K - P. \tag{3.15}$$

A common choice for a convolution layer takes and $P = K - S$, with the purpose of adjusting the spatial size by dividing with a controlled factor S:

$$N = S(M-1) + K - (K-S) \Rightarrow N = SM. \tag{3.16}$$

In **zero padding**, the padding contains extra zeros (figure 3.5). However padding can also introduce the periodic extension of a configuration. For Ising configurations, or any other configuration subject to periodic boundary conditions, **periodic padding** offers a convenient tool to intrinsically incorporate this periodicity in the model's architecture.

3.7 Cost functions

In the previous sections, a very extensive class of models was discussed, namely the neural networks. Apart from the model type, other key components for machine learning are the cost function and the minimization method to minimize this cost. This section will focus on cost functions and the next section will explain the minimization methods.

To make a performant model, it is very important to select the right cost function. The selection will depend upon the eventual task of the model. In the next subsections two types of supervised machine learning will be considered: classification and regression.

In supervised learning the cost $C(\mathbf{X}, f(\boldsymbol{\theta}))$ on dataset $\mathbf{X}$ containing N configurations $\mathbf{x}$, can be determined as the mean of the cost of these individual configurations:

$$C(f(\mathbf{X}|\boldsymbol{\theta})) = \langle C(f(\mathbf{x}|\boldsymbol{\theta}))\rangle_{\mathbf{x}\in\mathbf{X}}. \tag{3.17}$$

3.7.1 Classification: Cross-entropy cost

In supervised learning problems, we want to train a model to predict for a configuration $\mathbf{x}$ a certain feature $Y(\mathbf{x}) = \mathbf{y}$. Essential in classification is that the feature can only have a discrete set of outcomes, representing the possible categories to which the configuration can belong. If there are M possible categories C_i , where $i \in \{1, .., M\}$,

$$\mathbf{y} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_M \end{bmatrix} = \begin{bmatrix} P(\mathbf{x} \in C_1) \\ P(\mathbf{x} \in C_2) \\ \vdots \\ P(\mathbf{x} \in C_M) \end{bmatrix}, \tag{3.18}$$

where y_i represents the probability that $\mathbf{x}$ belongs to category C_i. $\mathbf{y}$ will always consist of $M - 1$ zeros and 1 one because every configuration belongs to exactly one category.

As said, the model f will provide a prediction $f(\mathbf{x}|\boldsymbol{\theta}) = \hat{\mathbf{y}}$ approximating the real feature $Y(\mathbf{x}) = \mathbf{y}$ as close as possible:

$$\hat{\mathbf{y}} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_M \end{bmatrix} = \begin{bmatrix} P_f(\mathbf{x} \in C_1) \\ P_f(\mathbf{x} \in C_2) \\ \vdots \\ P_f(\mathbf{x} \in C_M) \end{bmatrix}, \tag{3.19}$$

where $\hat{y}_i = P_f(\mathbf{x} \in C_i)$ represents the model's prediction for the probability that $\mathbf{x}$ belongs to category C_i.

To ensure that $\hat{\mathbf{y}}$ represents a valid probability distribution ($\sum_{i=1}^{M} \hat{y}_i = 1$ and $\hat{y}_i \in [0, 1]$), the final step of the model consists of normalizing the intermediate unrestricted result $\mathbf{z} \in \mathbb{R}^M$ into a probability distribution $\hat{\mathbf{y}} \in \mathbb{R}^M$ by means of the Softmax function:

$$\hat{\mathbf{y}} = \frac{\exp(\mathbf{z})}{\sum_{i=1}^{M} \exp(z_i)}. \tag{3.20}$$

Softmax transforms $\mathbf{z}$ to a probability distribution in a continuous way: in contrast to the real probabilities $P(\mathbf{x} \in C_i)$ which are either 0 or 1, the probabilities $P_f(\mathbf{x} \in C_i)$ can take any value in the interval [0,1]. This is necessary because the minimization method is dependent on gradients, requiring the model to be continuous.

If one would select a category out of this probability distribution $f(\mathbf{x}|\boldsymbol{\theta}) = \hat{\mathbf{y}}$, the probability that this category would correspond with the real category, C_j, to which $\mathbf{x}$ belongs is:

$$\begin{aligned} P(\text{correct prediction}|f(\mathbf{x}|\boldsymbol{\theta})) &= \left(\prod_{i=1, i\neq j}^{M} P_f(\neg C_i) \right) P_f(C_j) \\ &= \left(\prod_{i=1, i\neq j}^{M} (1 - \hat{y}_i) \right) \hat{y}_j \\ &= \prod_{i=1}^{M} \hat{y}_i^{y_i} (1 - \hat{y}_i)^{(1-y_i)} \end{aligned} \tag{3.21}$$

The model $f(\mathbf{x}|\boldsymbol{\theta}^*)$ with the optimal parameters maximizes this probability $P \in [0, 1]$. As a consequence this model also maximizes $\log(P) \in [-\infty, 0]$ and minimizes $-\log(P) \in [0, +\infty]$, which is an ideal candidate for the cost function:

$$\begin{aligned} C(f(\mathbf{x}|\boldsymbol{\theta})) &= -\log(P(\text{correct prediction}|f(\mathbf{x}|\boldsymbol{\theta})) \\ &= \sum_{i=1}^{M} -y_i \log(\hat{y}_i) - (1 - y_i) \log(1 - \hat{y}_i), \end{aligned} \tag{3.22}$$

known as the cross-entropy. The cost of the total dataset $\mathbf{X}$ containing $\mathbf{N}$ configurations $\mathbf{x}$, is then:

$$C(f(\mathbf{X}|\boldsymbol{\theta})) = \frac{1}{N} \sum_{i=1}^{N} C(f(\mathbf{x}_i|\boldsymbol{\theta})) = \frac{1}{N} \sum_{i=1}^{N} C(\mathbf{y}_i, \hat{\mathbf{y}}_i). \tag{3.23}$$

For binary classification ($M = 2$),the output is simplified to a scalar, $\hat{y}$, which represents $P_f(C_1)$, $\hat{y}_2$ is omitted since it is equal to $(1 - \hat{y})$. A Sigmoid activation ensures that $\hat{y}$ represents a valid probability $\in$ [0,1].

3.7.2 Regression: Mean Squared Error

In contrast with classification, regression models aim to predict continuous features $Y(\mathbf{x}) = y$. An example of such a continuous feature could be the energy of an Ising configuration[9]. The model provides a prediction $f(\mathbf{x}|\boldsymbol{\theta}) = \hat{y}$, approximating the real feature y as close as possible. Note that the output $\hat{y}$ is a scalar and no longer represents a probability, therefore, the output-layer no longer requires a softmax or sigmoid activation function.

The simplest and most popular cost function for a regression network is the mean-squared error between network predictions and the true value:

$$C(f(\mathbf{X}|\boldsymbol{\theta})) = \frac{1}{N}\sum_{i=1}^{N} C(f(\mathbf{x}_i|\boldsymbol{\theta})) = \frac{1}{N}\sum_{i=1}^{N} C(y_i, \hat{y}_i) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i)^2. \tag{3.24}$$

3.7.3 Regularization

In the introduction on machine learning (chapter 1.1), it was noted that a good model has the capacity to generalize: during training the cost C_{train} on the train-set $\mathbf{X}_{train}$ is minimized. However to evaluate the real performance of the model, the cost C_{test} on the test-set $\mathbf{X}_{test}$, is what counts. When C_{train} is significantly lower then C_{test}, the model has captured noise patterns that are specific for $\mathbf{X}_{train}$ and have nothing to do with global trends of the data. The model is **overfitted** to $\mathbf{X}_{train}$ and performs badly on new data.

One approach to address the problem of overfitting is to add an extra term to the cost function $C(f(\mathbf{X}|\boldsymbol{\theta}))$ in which large parameter-values are penalized. Excessive weights, often the result of noise on the training-data, can cause large, misguided variance of the output, which decreases the generalizing capacity of the model. L2-regularization or weight decay counters the tendency of the parameters to diverge to large numbers by adding the L2-norm of the parameters $\boldsymbol{\theta}$ as a penalty term to the cost function:

$$C_{tot}(f(\mathbf{X}|\boldsymbol{\theta})) = C_{original}(f(\mathbf{X}|\boldsymbol{\theta})) + \lambda||\boldsymbol{\theta}||_2^2, \tag{3.25}$$

where $\lambda > 0$ is the regularization hyperparameter. λ has to be chosen with care. If λ is too small, overfitting will still occur. If λ is too big, the model will be underfitted because the regularization-term then dominates the original cost-function $C_{original}(f(\mathbf{X}|\boldsymbol{\theta}))$, minimizing the parameters without performing the actual task.

[9] In reality, the energy of a finite Ising system is quantized, however the scale of quantization is so small it does not matter to the macroscopic problem.

[10][22]

3.8 Minimization Method

Machine learning consists of finding the optimal parameters $\boldsymbol{\theta}^*$ for which the cost is minimal:

$$\boldsymbol{\theta}^* = \text{argmin}_{\boldsymbol{\theta}}\{C(f(\mathbf{X}|\boldsymbol{\theta}))\}. \tag{3.26}$$

For a small data set with a simple polynomial model, optimization can be confined to a single step fit. However, chaotic and large datasets require extensive models such as neural networks. Once a certain limit in size and complexity of the model is exceeded, the single-step analytic approach is no longer possible[10] and iterative methods become the new standard. Remember that the same conclusions were made for the diagonalization to optimize PCA (chapter 2).

An initial guess determines $\boldsymbol{\theta}_0$. In every step i the minimization algorithm calculates the new values $\boldsymbol{\theta}_i$, eventually approximating the optimal parameters $\boldsymbol{\theta}^*$.

In this section, a very powerful and common class of iterative minimization methods is discussed: gradient descent and its variations.

In the remainder of this section, to ease notation, the cost-function $C(f(\mathbf{X}|\boldsymbol{\theta})$ can additionaly be written as $C(\boldsymbol{\theta})$ or $C(\mathbf{X}, \boldsymbol{\theta})$.

3.8.1 Gradient Descent

The basic idea behind gradient descent is to adjust the parameters in the direction of steepest descent in cost:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}_t), \tag{3.27}$$

where η is the learning rate, a hyperparameter controlling the size of the steps taken along the gradient-direction. Assuming an appropriate choice for this learning rate η, the parameters will eventually converge to a local minimum of the cost-function.

[10]The cost-function is a complicated, rugged, non-convex function in high-dimensional space with many local minima. [22]

For neural networks or, in general, every model containing a large number of parameters, the calculation of the gradient[11]

$$\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}_t) = \begin{bmatrix} \partial_{\theta_1} C(\boldsymbol{\theta}_t) \\ \partial_{\theta_2} C(\boldsymbol{\theta}_t) \\ \vdots \\ \partial_{\theta_N} C(\boldsymbol{\theta}_t) \end{bmatrix} \in \mathbb{R}^N \tag{3.28}$$

is responsible for a substantial amount of the computation time[12]. To calculate this gradient, a brute force derivation to every parameter is out of the question and a clever use of the chain rule is essential. The backpropagation algorithm efficiently incorporates these considerations for neural networks by exploiting their layered structure. Backpropagation is a standard component in most machine learning libraries and it will be briefly discussed in the next section.

To anticipate what alterations of the gradient descent method might be interesting to explore, a few remarks are listed:

1. The cost-function is a complicated hypersurface containing many local minima in which the gradient descent method might get stuck, resulting in poor performance. The barriers surrounding these local minima can be overcome by introducing stochasticity in the gradient descent algorithm.

2. What is an appropriate choice for the learning rate η? Choosing η too big can lead to overshooting the minimum, while a too small value might result in very slow convergence. Furthermore, it seems valid for η to change per iteration depending on how fast the gradient changes (i.e. the curvature of hypersurface $C(\boldsymbol{\theta})$).

3. To calculate the total gradient, the gradients of all the train-configurations are averaged (section 3.7):

$$\nabla_{\boldsymbol{\theta}} C(\mathbf{X}_{train}, \boldsymbol{\theta}) = \langle \nabla_{\boldsymbol{\theta}} C(\mathbf{x}, \boldsymbol{\theta}) \rangle_{\mathbf{x} \in \mathbf{X}_{train}}. \tag{3.29}$$

   However, a small subset of configurations might already give a good approximation of the gradient. This idea is worth exploring, since it could drastically decrease the computation-time per iteration.

[11] Note that both the cost and the gradient are only calculated for exactly one set of parameter-values per iteration: the parameters $\boldsymbol{\theta}_t$ of that instant.

[12] The calculation of the cost itself (forward propagation), complemented with the calculation of the gradient of the cost (backpropagation), forms the core of the training, being responsible for most of the computation time.

3.8.2 Backpropagation

As mentioned in the previous section, the backpropagation algorithm efficiently calculates the gradient by exploiting the layered structure of neural networks. The total network-function f can be written as a composition of single-layer-functions f^l (equation 3.4). In this section, we will focus on the gradient of a single configuration $\nabla_{\boldsymbol{\theta}} C(\mathbf{x}, \boldsymbol{\theta})$, since the total gradient is simply the average.

1. **Feedforward:** A configuration $\mathbf{x}$ is passed through the model $f(\boldsymbol{\theta})$, all weighted sums $\mathbf{z}^l$ and activations $\mathbf{y}^l$ are stored.

$$f(\mathbf{x}) = f^n \circ ... \circ f^2 \circ f^1(\mathbf{x}) = \mathbf{y}, \tag{3.30}$$

$$f^l(\mathbf{y}^{l-1}) = \sigma^l(\mathbf{W}^l \mathbf{y}^{l-1} + \mathbf{b}^l) = \sigma^l(\mathbf{z}^l) = \mathbf{y}^l, \tag{3.31}$$

   with $l \in \{1, .., n\}$, $\mathbf{y}^0 = \mathbf{x}$ and $\mathbf{y}^n = \mathbf{y}$. All parameters $\in \boldsymbol{\theta}$ are written down as either w^l_{jk}, the weight of the link between node k at layer $l-1$ to node j at layer l, or b^l_j, the bias of neuron j at layer l.

2. From the output $\mathbf{y}$ the **cost** $C = C(f(\mathbf{x}|\boldsymbol{\theta}))$ is calculated.

3. **Backpropagation**: We propagate back through the model, layer by layer. For every layer l, the error function $\boldsymbol{\nabla}_{\mathbf{z}^l} C$ is calculated by recursion:

$$\boldsymbol{\nabla}_{\mathbf{z}^{l+1}} C \xrightarrow{\mathbf{W}^{l+1}, \mathbf{z}^l} \boldsymbol{\nabla}_{\mathbf{z}^l} C, \tag{3.32}$$

   using the following formula:

$$\begin{aligned} (\boldsymbol{\nabla}_{\mathbf{z}^l} C)_k &= \frac{\partial C}{\partial z^l_k} \\ &= \boldsymbol{\nabla}_{\mathbf{z}^{l+1}} C \cdot \frac{\partial \mathbf{z}^{l+1}}{\partial z^l_k} \\ &= (\boldsymbol{\nabla}_{\mathbf{z}^{l+1}} C \cdot \mathbf{W}^{l+1}_{*k})\ \sigma^{l\prime}(z^l_k). \end{aligned} \tag{3.33}$$

4. **Calculate the gradient:**

$$\frac{\partial C}{\partial w^l_{jk}} = (\boldsymbol{\nabla}_{\mathbf{z}^l} C)_j\ y^{l-1}_k \tag{3.34}$$

   and

$$\frac{\partial C}{\partial b^l_j} = (\boldsymbol{\nabla}_{\mathbf{z}^l} C)_j \tag{3.35}$$

3.8.3 Variations on Gradient Descent

Stochastic Gradient Descent

A variant on the Standard Gradient Descent method is Stochastic Gradient Descent. Where the gradient $\nabla_{\boldsymbol{\theta}} C(\mathbf{X}_{train}, \boldsymbol{\theta})$, evaluated over the total train-set $\mathbf{X}_{train}$, is approximated with a *stochastic* gradient evaluated over a small subset or batch $\mathbf{B} \subset \mathbf{X}_{train}$:

$$\nabla_{\boldsymbol{\theta}} C(\mathbf{X}_{train}, \boldsymbol{\theta}) = \langle \nabla_{\boldsymbol{\theta}} C(\mathbf{x}, \boldsymbol{\theta}) \rangle_{\mathbf{x} \in \mathbf{X}_{train}} \rightarrow \nabla_{\boldsymbol{\theta}} C(\mathbf{B}, \boldsymbol{\theta}) = \langle \nabla_{\boldsymbol{\theta}} C(\mathbf{x}, \boldsymbol{\theta}) \rangle_{\mathbf{x} \in \mathbf{B}}. \quad (3.36)$$

Every iteration is performed with a stochastic gradient. The total train-set is subdivided into N_B batches. Therefore, one epoch, i.e. one evaluation of the total train-set $\mathbf{X}_{train}$, consists of N_B iterations.

The calculations are sped up significantly since the cost and gradient are approximately determined, requiring less configurations. Another advantage is given by the fact that the gradient is not determined perfectly, there is a component of stochasticity. This decreases the chance to get stuck in a local mini bsubsectionMomentum Further alterations to the SGD method include the use of a memory term[22][31]:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \mathbf{m}_t, \quad (3.37)$$

with

$$\mathbf{m}_t = \gamma \mathbf{m}_{t-1} + \eta \nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}_t), \quad (3.38)$$

where $0 \leq \gamma \leq 1$. The movement in the parameter-space $\mathbf{m}_t$ is no longer completely determined by the gradient at time t: $\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}_t)$, but also by earlier gradients through $\mathbf{m}_{t-1}$. The momentum $\mathbf{m}_t$ is calculated as an exponentially weighted, running average of recent gradients, where the hyper-parameter γ determines the memory's characteristic time scale: $\frac{1}{1-\gamma}$.

This approach has several advantages: In directions with persistent but small gradients (the cost-function is shallow, low curvature), the gradient $\nabla_{\boldsymbol{\theta}} C(\boldsymbol{\theta}_t)$, and earlier gradients incorporated in $\mathbf{m}_{t-1}$ are oriented in the same direction and reinforce each other. The total momentum gains weight and therefore the algorithm can proceed quicker. In areas with rapidly changing gradients (narrow, high curvature), the gradient will be partially cancelled by the earlier gradients because it is oriented in a different direction. The total momentum becomes smaller, therefore the algorithm proceeds in a more careful way.

Second moment

As explained in section 3.8.1 on the GD method, finding an appropriate value for the learning rate η is crucial. Furthermore, it can be interesting to tune η in function of time, depending on the curvature of the cost-function.

Up until this point, the step-size $|\Delta\boldsymbol{\theta}|$ that is taken each iteration, is proportional to (a running average of) the steepness of the cost-function $C(\boldsymbol{\theta})$:

$$\Delta\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_{t+1} - \boldsymbol{\theta}_t = \mathbf{m}_t = \gamma\mathbf{m}_{t-1} + \eta\nabla_{\boldsymbol{\theta}}C(\boldsymbol{\theta}_t), \tag{3.39}$$

However, more important than steepness (1st derivative), is curvature (2nd derivative). The step-size $|\Delta\boldsymbol{\theta}|$ should be larger in shallow areas , i.e. areas where the gradient varies slowly. Whereas in unpredictable, narrow areas, with quickly changing gradients, one should proceed more carefully, resulting in a smaller step-size $|\Delta\boldsymbol{\theta}|$.

To address these requirements, one might think to calculate the curvature similar to how the gradient was calculated. However, based on the cost-benefit ratio, this could never be an improvement, since the computational cost is very high.

In reality, estimations of the curvature are made by keeping track on the variation of the gradient over time.

Let's discuss Adam[22][31], an alteration of SGD which incorporates this strategy[13]. To alleviate notation, the gradient will be written as follows:

$$\mathbf{g}_t = \nabla_{\boldsymbol{\theta}}C(\boldsymbol{\theta}_t). \tag{3.40}$$

The running average of the gradient, is defined in a way that slightly deviates from the previous section:

$$\mathbf{m}_t = \beta_1\mathbf{m}_{t-1} + (1-\beta_1)\mathbf{g}_t \tag{3.41}$$

$$\hat{\mathbf{m}}_t = \frac{\mathbf{m}_t}{1-\beta_1}. \tag{3.42}$$

$\mathbf{m}$ is the first moment of the gradient. In addition, we can also keep track of the second moment $\mathbf{s}$:

$$\mathbf{s}_t = \beta_2\mathbf{s}_{t-1} + (1-\beta_2)\mathbf{g}_t^2 \tag{3.43}$$

[13] Apart from Momentum or Adam, multiple other variations on gradient descent have been developed. E.g. Nesterov accelerated gradient (NAG), which is an extension to Momentum, RMS prop, which is similar to Adam, Adagrad and Adadelta, which are also similar to Adam without the use of momentum, etc. [22][39][31]

$$\hat{\mathbf{s}}_t = \frac{\mathbf{s}_t}{1-\beta_2}, \tag{3.44}$$

β_1 and β_2 determine the characteristic timescales of the first and second moment and are typically given the values 0.9 and 0.99 respectively. The variation on the gradient, which is an indication for the curvature of the cost-function, can be related to the first and second moment by the following expression:

$$\mathbf{var}_t^2 = \hat{\mathbf{s}}_t - \hat{\mathbf{m}}_t^2. \tag{3.45}$$

The update rule for Adam is given by:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \frac{\hat{\mathbf{m}}_t}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon}, \tag{3.46}$$

where η is the learning rate and ϵ is a very small regularization constant (order 10^{-8}) to prevent division by zero. The division in this equation, has to be seen as an element-wise operation. Let's take a closer look for a single parameter $\theta \in \boldsymbol{\theta}$:

$$\theta_{t+1} = \theta_t - \eta_t \hat{m}_t, \tag{3.47}$$

with

$$\eta_t = \frac{\eta}{\sqrt{\hat{s}_t} + \epsilon} = \frac{\eta}{\sqrt{\mathrm{var}_t^2 + \hat{m}_t^2} + \epsilon}. \tag{3.48}$$

Equation (3.47) reminds of the simple SGD method, where the fixed learning rate η has been replaced by η_t, which is parameter-dependent and varies per iteration. What determines the eventual value of η_t? Two limiting cases are examined:

1. If the derivative $\partial_\theta C(\boldsymbol{\theta})$ is only slowly changing (i.e. the cost-function is shallow in the θ-direction in parameterspace), then:

   $$\sqrt{\mathrm{var}_t^2 + \hat{m}_t^2} \approx \hat{m}_t. \tag{3.49}$$

   By consequence we find that $\eta_t = \eta$. The update rule becomes:

   $$\theta_{t+1} = \theta_t - \eta, \tag{3.50}$$

   taking the largest possible step-size determined by the maximal learning rate η.

2. If the derivative changes rapidly (narrow in the θ-direction in parameterspace), the following applies:

   $$\sqrt{\mathrm{var}_t^2 + \hat{m}_t^2} \approx \mathrm{var}_t. \tag{3.51}$$

The update rule becomes:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}}{\text{var}_t}, \tag{3.52}$$

meaning that the step-size is proportional to the signal to noise ratio. In this way, the algorithm proceeds more careful in highly curved directions.

In general, Adam makes sure that the step-size is automatically adapted by the local curvature of the cost-function.

4 Neural Networks on classical spin systems

In this chapter, the performance of neural networks for extracting the features of classical spin systems are evaluated. Firstly, these networks are trained to classify Ising configurations into either the ferro- or the paramagnetic phase. Subsequently, the same networks are trained to predict the energy and magnetization of Ising configurations through regression.

4.1 Classification

The classification network takes an Ising configuration $\mathbf{x}$ as input and delivers a single output value $\hat{y}$, representing the predicted probability[1] that the configuration $\mathbf{x}$ belongs to the ferromagnetic phase. The desired output is represented by a binary label y, which is 1 for a ferromagnetic configuration and 0 for the a para magnetic configuration. For classification problems, the performance of a model is measured by the binary cross-entropy cost function[22]:

$$C(\mathbf{x}, \boldsymbol{\theta}) = -y \log(\hat{y}) - (1 - y) \log(1 - \hat{y}), \tag{4.1}$$

The total cost is always calculated for a set of samples $\mathbf{X}$

$$C(\mathbf{X}, \boldsymbol{\theta}) = \langle C(\mathbf{x}, \boldsymbol{\theta}) \rangle_{\mathbf{x} \in \mathbf{X}},$$

where $\mathbf{X}$ can be either a batch $\mathbf{B} \subset \mathbf{X}_{train}$ to determine C_{train} during optimization, or the testing sets $\mathbf{X}_{T,test}$, $\mathbf{X}_{test}$, to determine the final loss (per temperature) $C_{T,test}$, C_{test} for a set of configurations never encountered during training.

4.1.1 General performance of the model

In this chapter, it will become clear that the training of neural networks requires making many choices. What is an adequate architecture for the network? How many configurations should the training set contain? Which minimization method should be used and how should their corresponding hyper parameters be optimized?

[1] to ensure that $\hat{y} \in [0, 1]$ the output neuron contains a sigmoid activation function [22].

Prior to the answering of these questions, this section establishes the results that are shared among all trained classifiers.

Characteristic peak in the loss function

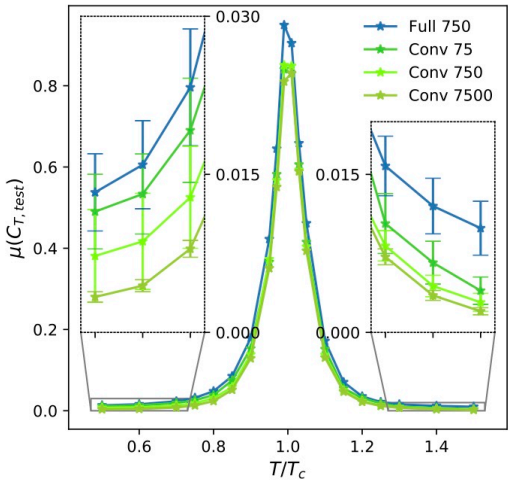


Fig. 4.1: Performance of neural networks on the classification of the Ising configurations. The average cost on the testing set $\mu(C_{T,test})$ is displayed as a function of temperature for four types of trained models with differing architecture: fully connected (blue) or convolutional (green) and training set size (shades of green). Each type of training was repeated 25 times. In the insets, the curves are zoomed in and the standard deviation is added.

Figure 4.1 shows the average cost on the test set $\mu(C_{T,test})$ as a function of temperature for four differing types of training:

- 1 fully connected neural network (26721 parameters) is trained with a training set containing 750 configurations per temperature.

- 3 convolutional neural networks with the same architecture (771 parameters) are trained with training sets of varying sizes: $75, 750$ and 7500 configurations per temperature.

The test set $\mathbf{X}_{test}$ is an assembly of the sets per temperature, $\mathbf{X}_{T,test}$, each containing 250 configurations.

The predominant observation is the characteristic peak in the cost function around the critical temperature, T_c, which retains approximately the same shape and size regardless of the type of training. The cost peak indicates the failure of the network to correctly classify the configurations in a small temperature range surrounding the critical temperature T_c. This resembles a phenomenon that was described in the introductionary chapter:

For finite lattice sizes, $|m|$ and e display strong fluctuations around transition temperature. These fluctuations imply that, in contrast to the infinite lattice, for a certain temperature range, ΔT_L, around the critical temperature T_c, $|m|$ and e are no longer capable of providing a correct prediction of the phase of a configuration. This range of uncertainty increases for decreasing lattice size L.

This observation seems to indicate that neural networks base their classification on features that, similar to $|m|$ and e, display strong fluctuations around transition. In reference [8], it is shown that the width of cost peak decreases with increasing lattice size L.[2] which further strengthens this theory. From these observations, we conclude that it is plausible that there is a lower boundary for the cost function which completely depends on the physics of the phase transition and is independent of the specifics of the chosen model.

Evolution of the training

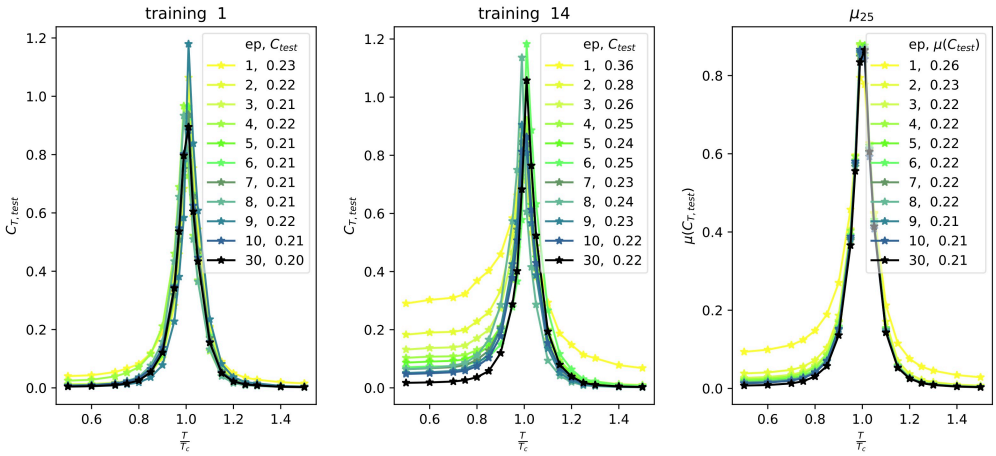


Fig. 4.2: Evolution of $C_{T,test}$ per epoch during training. The left and middle graph show the evolution of one training, the right graph shows the average evolution of 25 trainings. (**Specifications of training:** modeltype: convolutional, optimizer: SGD, batch size: 20, learning rate : 10^{-2}, # train (test) conf: 750(250), patience: 5)

In figure 4.2, the cross-entropy loss is plotted as a function of the temperature for consecutive epochs. The left and middle graph show the evolution for single trainings, the right graph shows the average evolution of 25 separate trainings.

On average, the model took 22.2 epochs to converge. But the convergence time varies strongly from training to training; the standard deviation was determined to be 8.3 epochs. The difference between training 1 and training 14, reflects this strong variation (left and middle panel of figure 4.2). It should be noted that the

[2] A finite size scaling analysis was done, correctly inferring information on the infinite lattice model.

specific amount of epochs for convergence depends on the specific architecture of the network and the training specifications. However, qualitatively the same trends were found regardless of the training specifications.

For training 14 (center of figure 4.2), the slow convergence of the low-temperature costs is striking. This phenomenon was regularly encountered: for all five model types and also for different optimizer types (namely Adam instead of SGD). The contribution of these asymmetric training visibly influences the average evolution (figure 4.2).

4.1.2 Model architectures

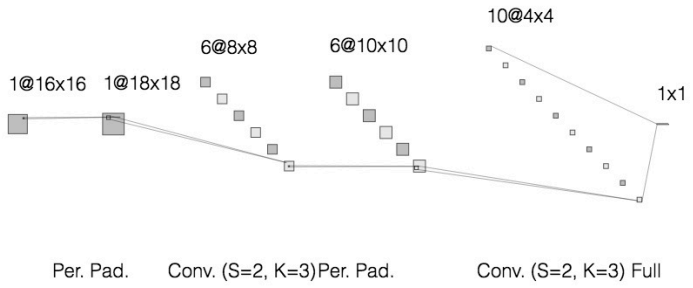


Fig. 4.3: Structure of ConvK3PP: a convolutional neural network with kernel size $K = 3$ and periodic padding. S indicates the stride of the convolutions.

To test the influence of the architecture of the neural networks on the performance, five model types were introduced:

- One fully connected neural network (FCNN).
- Four convolutional neural networks (CNN):
  - a CNN with kernel size $K = 5$.
  - a CNN with $K = 5$ and periodic padding (PP).
  - a CNN with $K = 3$.
  - a CNN with $K = 3$ and PP.

They were repectively named: Full, ConvK5, ConvK5PP, ConvK3 and ConvK3PP. In figure 4.4, the performances of these 5 model types are compared. For each architecture type 25 separate trainings were performed. In the left panel, cost as a

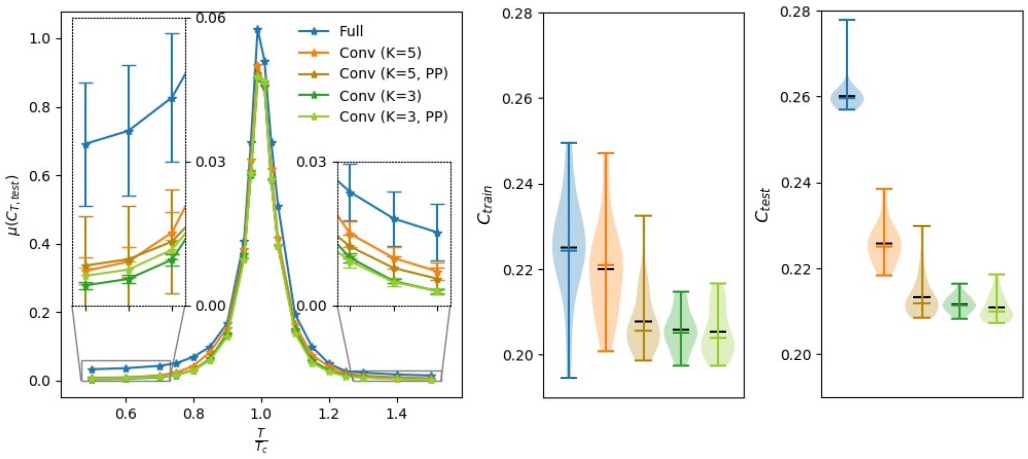


Fig. 4.4: In this figure, the performances of 5 model architectures are compared: one fully connected neural network (FCNN) and 4 convolutional neural networks (CNN) (kernel size K ∈ {5,3} and without/with periodic padding). For each architecture type 25 separate trainings were performed. (a) The average cost as a function of temperature $\mu(C_{T,test})$. The insets enlarge the low-cost regions, where the errorbars indicate the standard deviation of the cost per temperature $\sigma(C_{T,test})$. (b) distribution of the cost C_{train}. (c) distribution of the cost C_{test}. (**Specifications of training:** optimizer: SGD, batch size: 64, learning rate: 10^{-2}, # train (test) conf: 750(250), patience: 5)

function of temperature is shown, and we see again the characteristic peak that was described in the previous section, which has approximately the same shape and size for all architectures. If calculated in terms of correctly classified configurations, the performance is approximately 90% for all model types (table 4.1).

However, the insets show that there are still notable differences for the different architectures. These differences are more visible in the center and right panel which show the distribution of the cost on the training and testing set, C_{train} and C_{test}, for each model type:

1. For the fully connected network, the distribution of C_{test} lies significantly higher than the distributions of C_{train}, indicating overfitting. For convolutional networks too, overfitting occurs, however for CNNs the overfitting is far less

Tab. 4.1: Comparison of the performance for various neural network architectures.

modeltype	correct predictions: μ (σ) [%]	C_{test}: μ ($\sigma[10^{-3}]$) (σ_r [%])
Full	89.4 (0.10)	0.260 (3.80) (1.46)
Conv (K = 5)	90.2 (0.17)	0.226 (4.69) (2.08)
Conv (K = 5, PP)	90.6 (0.12)	0.213 (4.74) (2.22)
Conv (K = 3)	90.5 (0.09)	0.212 (1.90) (0.90)
Conv (K = 3, PP)	92.9 (0.09)	0.211 (2.94) (1.40)

severe. Because of the restrictions on the architecture of CNN, significantly less parameters are needed for a model that has similar performance (table 4.2). High dimensional parameter spaces have more freedom to specialize on noise of the training set. In general, FCNN performs worse than CNN, comparing C_{test} to C_{train} shows that this is mostly due to overfitting. It can be concluded that models which implement restrictions based on information from the theoretical model or properties of the dataset the model has better performance, mainly because less parameters are needed.

2. The influence of the kernel size was not yet anticipated in the previous chapter. Roughly stated, the kernel size determines how many neighbor information is taken into account when processing the layer. $K = 5$ evaluates two layers of neighbors (blue border in figure 4.5), whereas $K = 3$ evaluates only one layer of neighbors (green border). If one examines the structure of the Ising hamiltonian (for $H = 0$)

$$E = -J \sum_{i,j=nn(i)}^{N} s_i s_j, \tag{4.2}$$

only the nearest neighbors are considered. Therefore, the use of kernel size $K = 3$, is a physically well-chosen, further restriction on the model, similar to the transition from FCNN to CNN.

Remark that, even though the hamiltonian only considers nearest neighbors, the scale of correlations is certainly not restricted to the nearest neighbors. Remember that at T_c, the correlation length equals the lattice size L (chapter 1.2.1). The choice of kernel size $K = 3$, examines only nearest neighbors in the convolution. This does however not imply that the total model will fail to recover large scale correlations; as explained in section 3.6, consecutive convolution layers focus on correlations at increasing scales.

This larger restriction on the model was expected to further reduce the models susceptibility to overfitting, however, looking at figure 4.4 this effect is not

Tab. 4.2: Dimension of the parameter space for various neural network architectures.

modeltype	# parameters
Full	26 721
Conv (K = 5)	3 287
Conv (K = 5, PP)	3 287
Conv (K = 3)	771
Conv (K = 3, PP)	771

clearly visible. Still, the models with $K = 3$ have better average performance and less variance in the performance than models with $K = 5$.

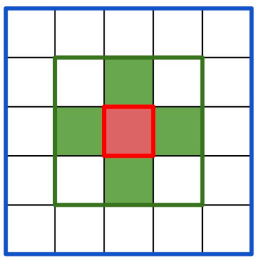


Fig. 4.5: Influence range of a kernel. Blue border: kernel size K = 5, two layers of neighbors are evaluated. Green border: K = 3, one layer of neighbors is evaluated. Green fill: the nearest neighbors which are evaluated by the hamiltonian when calculating the energy of the central spin (red).

3. The influence of periodic padding incorporates the periodic boundary conditions of the Ising configurations in the networks architecture. For $K = 5$, the model with periodic padding performs significantly better. For $K = 3$ however, there is not much improvement. Introducing periodic padding also seems to be linked with higher variance on the performance.

4.1.3 Size of the training set

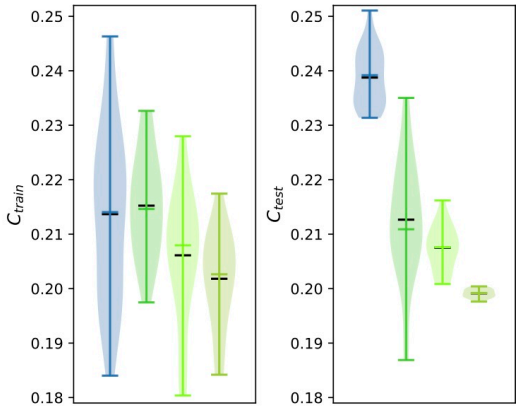


Fig. 4.6: Performance of classifier for four types of models: a fully connected neural network (blue) and three convolutional networks with various training set sizes (shades of green). Distribution of the total cost on both the training (left) and testing (right) configurations: C_{train} and C_{test}

In this section, the influence of the training set size on the performance of the model is investigated.

In figures 4.6 and 4.7 the influence of varying training set size is shown for training sets containing respectively 75, 750 and 7500 configurations per temperature. The best performing architecture, as found in the previous section, was used (ConvK3PP).

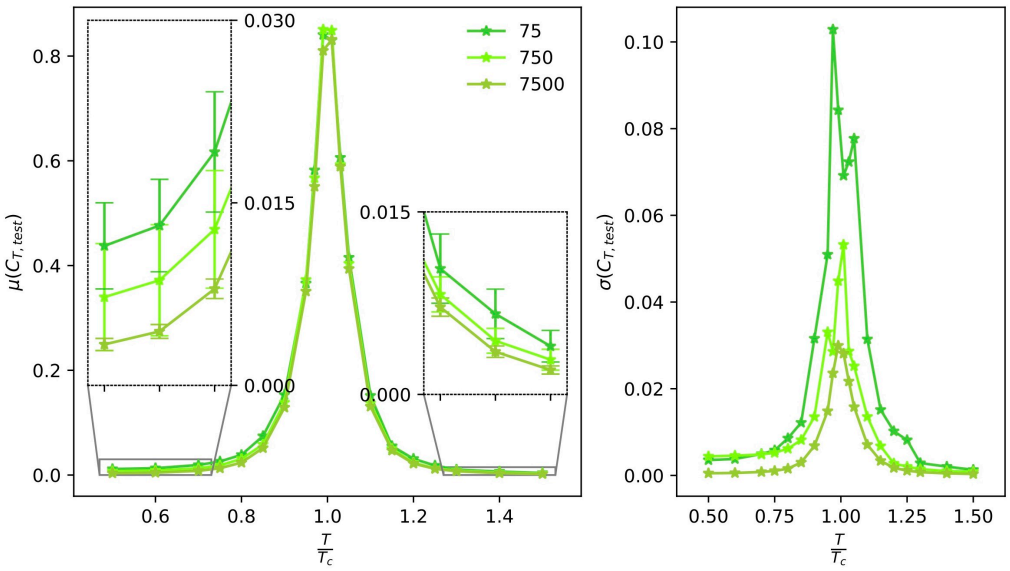


Fig. 4.7: Performance of classifier for varying sizes of the training set. Left: The average cost as a function of temperature $\mu(C_{T,test})$. The insets enlarge the low-cost regions, where the error bars indicate the standard deviation of the cost per temperature $\sigma(C_{T,test})$. Right: separate plot of the standard deviation of the cost per temperature $\sigma(C_{T,test})$. (**Specifications of training:** modeltype: CNNK3PP, optimizer: SGD, batch size: 20, learning rate : 10^{-2}, # train (test) conf: [75(250), 750(250), 7500(250)], patience: 7)

In figure 4.6, the performance of the fully connected neural network is added as an additional reference.

Figure 4.6 shows that both the average and the standard deviation of the cost decrease for an increasing number of training samples. The model is less biased by noise on the training data, because a larger data set gives a more complete representation of the canonical ensemble. Figure 4.7 shows the cost and standard deviation per temperature, indicating that this decrease happens at all temperatures.

The standard deviation per temperature, was not discussed before (left panel of figure 4.7). In order to interpret this curve, it should be noted that the total standard deviation on the performance $\sigma(C_{test})$ cannot be simply deduced from the

contributions per temperature $\sigma(C_{T,test})$, since these are correlated. The covariances between different temperatures should be taken into account (figure 4.8):

$$\begin{aligned}\sigma(C_{test})^2 &= \text{var}(C_{test}) \\ &= \text{var}(\frac{1}{N_T}\sum_i C_{T_i,test}) \\ &= \frac{1}{N_T^2}\sum_i\sum_j \text{covar}(C_{T_i,test}, C_{T_j,test}) \\ &= \frac{1}{N_T^2}\sum_i \left[\sigma(C_{T_i,test})^2 + \sum_{j\neq i}\text{covar}(C_{T_i,test}, C_{T_j,test})\right] \\ &= \frac{1}{N_T^2}\sum_i \sigma'(C_{T_i,test})^2,\end{aligned} \tag{4.3}$$

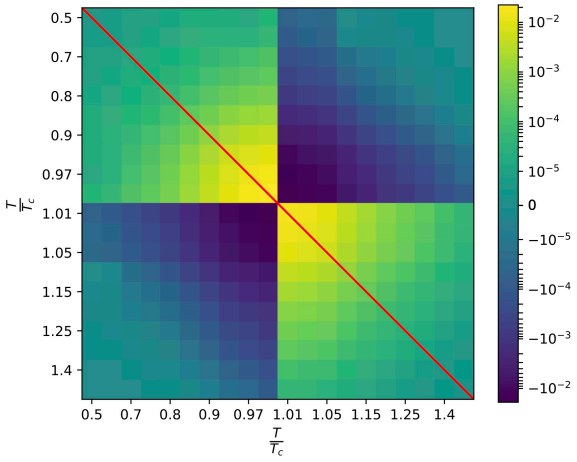
where N_T is the number of considered temperatures and σ' is the notation for a pseudo standard deviation that represents the variation at a certain temperature including the coupled variation with other temperatures. In contrast to σ, σ' contains all information to calculate the variation on the total cost. The covariance matrix of $C_{T,test}$ indicates that differences between separate trainings mainly occur in the form of a small translation of the cost function $C_{T,test}$ to higher or lower temperatures (figure 4.8). In principle[3], a mere translation of the cost function causes no variation in the total cost C_{test}, this illustrates the importance of the covariance terms in equation (4.3) to counter the $\sigma(C_{T,test})$ contributions. However, since the predicted transition temperature is shifted with respect to the real transition temperature T_c, a shift in the cost function will, in general, also cause an increase of the cost function and therefore the total cost will vary after all.

The benefits of using a larger training set are opposed by the increased amount of calculation time that is needed to complete a training. Table 4.3 contains average training times of models with varying set-sizes, all trainings were done under similar circumstances on the same GPU.

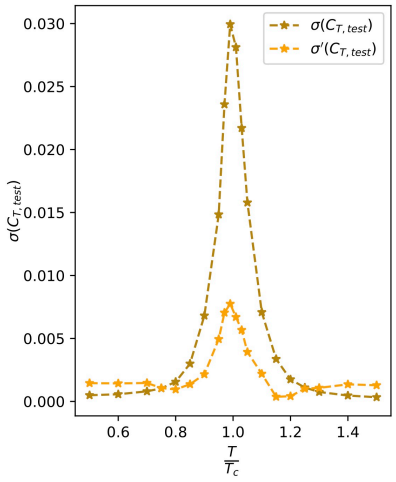
[3] This is only true for a data set consisting of data sets at uniformly distributed temperatures (which is not the case in this work).

Tab. 4.3: Average training time per model for various sizes of training sets.

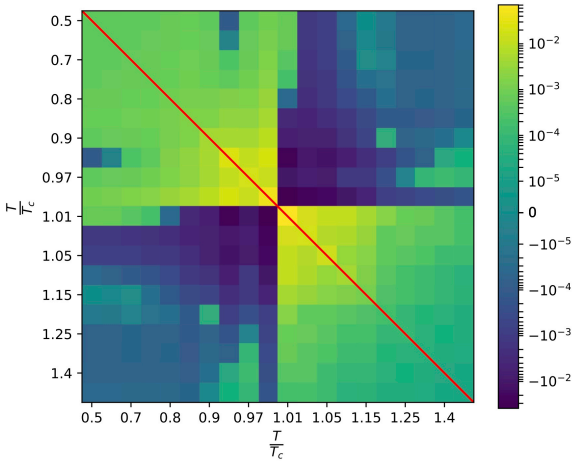
size training set	# epochs	time/epoch [s]	total time [s]
75	29.4	1.70	49.8
750	22.2	2.44	54.1
7500	14.0	23.78	332.0



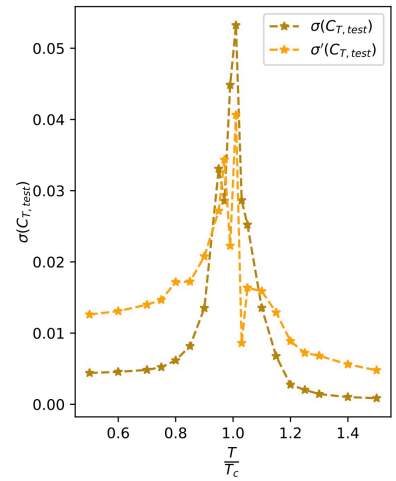
(a) Covariance matrix of $C_{T,test}$ (# training conf: 7500)



(b) $\sigma(C_{T,test})$ and $\sigma'(C_{T,test})$ (# training conf: 7500)



(c) Covariance matrix of $C_{T,test}$ (# training conf: 750)



(d) $\sigma(C_{T,test})$ and $\sigma'(C_{T,test})$ (# training conf: 750)

Fig. 4.8: Left: Covariance matrix (the red line indicates $\text{var}(C_{T,test}) = \sigma(C_{T,test})^2$). Right: standard deviation $\sigma(C_{T,test})$ and pseudo-standard deviation $\sigma'(C_{T,test})$ (equation (4.3)). Upper: 7500 training configurations. Lower: 750 training configurations. The graphs show that, in the transition region, most cost variation is countered by negative covariations with other temperatures ($\sigma > \sigma'$). This phenomenon can be linked with small translation of the cost function $C_{T,test}$ to higher or lower temperatures. The effect is more outspoken for larger training sets. At temperatures further away from transition, the cost variation is strenghtened with positive covariations with other temperatures ($\sigma < \sigma'$). This phenomenon can be linked with an overall higher or lower cost function. This effect is more outspoken for smaller training sets. (**Specifications of training:** modeltype: ConvK3PP, optimizer: SGD, batch size: 20, learning rate : 10^{-2}, # train (test) conf: [7500(250), 750(250)] patience: 5)

4.1.4 Interpretation of the convolutional model

In section 3.5, it was discussed that the steering of a model enhances the possibility to gain insight in the network. In this section, we will put this claim to the test.

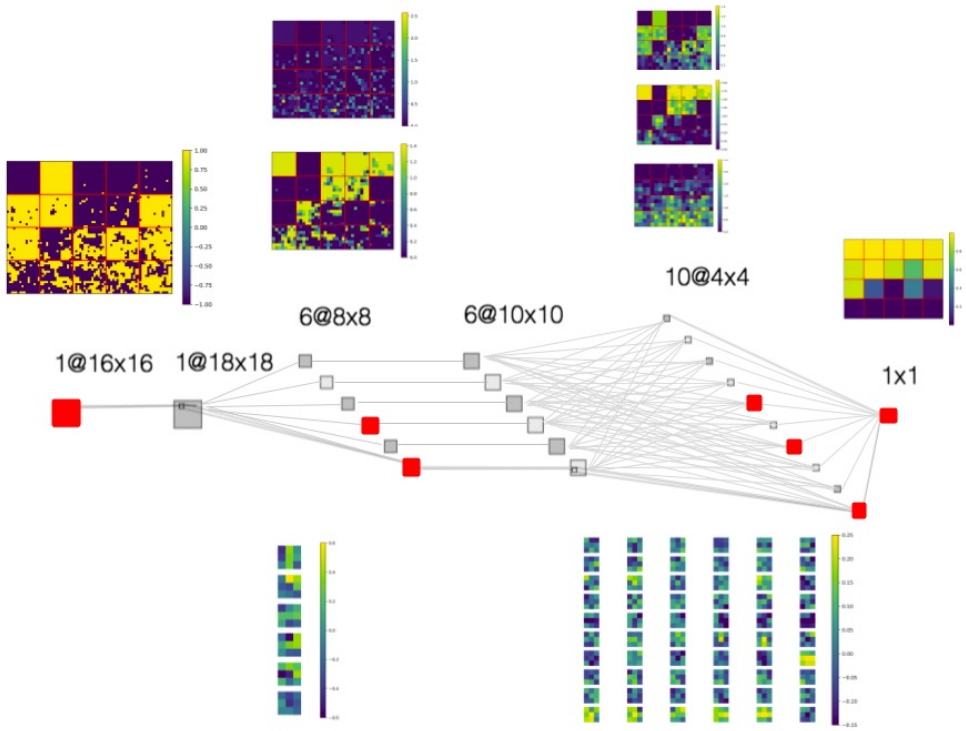
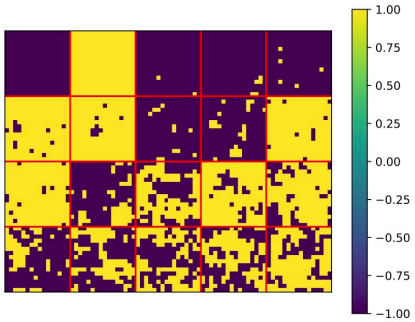


Fig. 4.9: Overview of the trained network. Central, the architecture of the network is shown. The network has 2 convolution layers, both with kernel size $K = 3$ and stride $S = 2$. Below each convolution layer, the corresponding trained kernels are shown. Above the architecture of the network, several intermediate activations for 20 input configurations, drawn from the testing set $\mathbf{X}_{test}$ for each temperature, are depicted. The displayed activations are indicated in red on the network's architecture (**Specifications of training:** modeltype: CNNK3PP, optimizer: SGD, batch size: 20, learning rate: $10^{-}2$, # train(test) conf: 750(250), patience: 5)

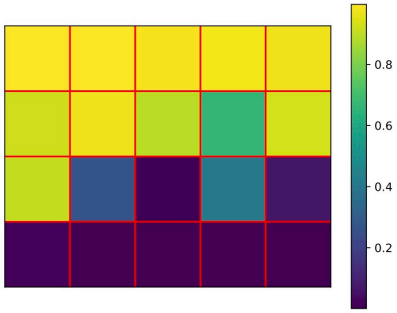
Section 3.6 introduced convolutional networks, which can be considered steered models in comparison with fully connected models. Due to locality and homogeneity considerations the number of weights were reduced and ordered in kernels. An attempt is made to understand the inner workings of a trained convolutional network by examining its kernels and intermediate activations.

Figure 4.9 gives an overview of the trained network. Central, the architecture of the network is shown. The network has 2 convolution layers. Below each convolution layer, the corresponding trained kernels are shown. Above the architecture of the network, several intermediate activations for 20 input configurations, one for each temperature included in the data set, are depicted.

All activations are given a label to summarize their most important characteristics:



(a) Input configurations, type $O(+)_1$



(b) Corresponding scalar outputs, type O_{256}

Fig. 4.10: (a) 20 Ising configurations, one for each temperature, are drawn from the testing set $\mathbf{X}_{test}$, with the temperature increasing from top left to bottom right. (b) the 20 outputs generated by feeding these Ising configurations to the network in figure 4.9.

- $O(+)$: Activations with high values in **ordered** areas with **positive** net magnetization and low values in either ordered areas with negative net magnetization or disordered areas (that have net magnetization zero).

- $O(-)$: Activations with high values in **ordered** areas with **negative** net magnetization and low values in either ordered areas with positive net magnetization or disordered areas (that have net magnetization zero).

- O: Activations with high values in **ordered** areas with non-zero net magnetization and low values in disordered areas.

- D: Activations with high values in **disordered** areas with zero net magnetization and low values in ordered areas.

The meaning of the word *area* in these explanations requires further elaboration; it indicates the scale on which the spin configurations are evaluated. This scale grows layer by layer. E.g. the input layer *area* includes only one spin, the first convolution layer includes nine[4], the second convolution layer includes 25 and finally, for the output, the area encompasses the complete lattice ($256 = 16^2$ spins). The scale on which the activation is evaluated is added as a subscript: e.g. the input configurations have label $O(+)_1$.

[4]From the perspective of dimensionality reduction of the input, every four input spins correspond to one value in the activations of the first convolutional layer (based upon the stride $S \times S, S = 2$). However, the total number of spins, that determine a single value in each activation, is in fact 9 (based upon the size of the kernel: $K \times K, K = 3$). This implies that the regions of influence are overlapping.

Since order over the complete lattice determines the ferromagnetic phase, the scalar output of the trained network is required to have label O_{256}.

What mechanisms translate the input configuration to the eventual output? To find the answer to this question, the kernels are investigated. In section 3.6, it was explained that the activations of every layer are calculated by taking the convolution of the activations of the previous layers with corresponding kernels:

$$\mathbf{Y}^{l,j} = \sigma(\sum_{k=1}^{D_{l-1}} \mathbf{W}^{l,j,k} *_S \mathbf{Y}^{l-1,k} + b^j \ \mathbf{1}), \tag{4.4}$$

with $j \in \{1, .., D_l\}$, D_l is the number of channels, present in layer l, $\mathbf{Y}^{l,j} \in \mathbb{R}^{N_l \times N_l}$ represents the activation of layer l channel j, $\mathbf{W}^{l,j,k} \in \mathbb{R}^{K \times K}$ represents the kernel, with kernel size K, from channel k in layer $l-1$ to channel j in layer l, $*_S$ represents a convolution with stride S.

The kernel size of the model under consideration is 3, therefore, every kernel contains 9 independent values. To stay focussed on the main principles however, all kernels are summarized in two values that capture their most important characteristics. The average $\mu = \langle \mathbf{W} \rangle$ and the contrast $\Delta = \max(\mathbf{W}) - \min(\mathbf{W})$. Using these two values, the main results of the convolution of each kernel with an input can be described:

- $\mu > 0$ and $|\mu| > \mu^*$ (positive, P): on average, the signs of the input will be retained.
- $\mu < 0$ and $|\mu| > \mu^*$ (negative, N): on average, the signs of the input will be switched.
- $\mu \approx 0$ or $|\mu| < \mu^*$ (zero, Z): the average value of the input will be shifted to zero

and

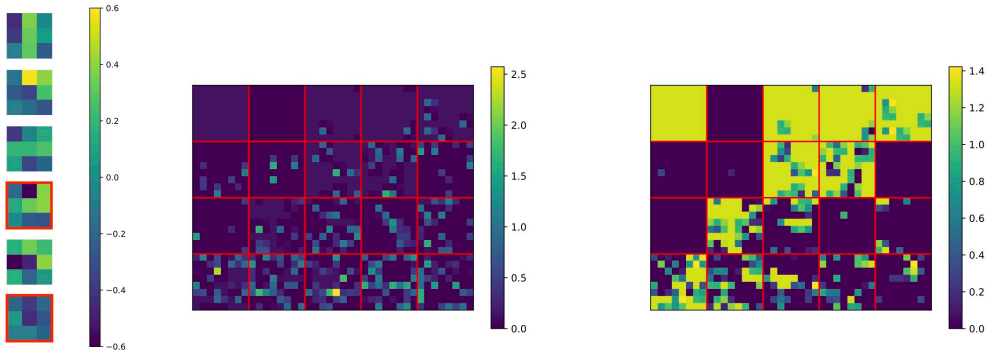
- $\Delta > \Delta^*$ (high contrast, HC): the variations in the input will be highlighted.
- $\Delta < \Delta^*$ (low contrast, LC): the variations in the input will be smoothened.

μ^* and Δ^* are the boundaries that decide how a kernel should be classified, they are determined experimentally based on the average sizes $|\mu|$ and Δ of the kernels. It should be kept in mind that every kernel is complemented with an activation function σ. For the network under consideration, this activation function is the

rectified linear unit ReLU, which retains all positive values and projects all negative values to zero.

First convolutional layer

In table 4.4 the average μ and contrast Δ for every kernel of the first convolutional layer are summed. The first five kernels have a low average value ($|\mu| < 0.08$) and high contrast ($\Delta > 0.5$), the sixth kernel has a negative average ($\mu < 0$ and $|\mu| > 0.08$) and a low contrast ($\Delta < 0.5$). The boundaries $\mu^* = 0.08$ and $\Delta^* = 0.5$ were determined experimentally[5].



(a) (b) (c)

Fig. 4.11: In (a), the kernels of the first convolutional layer are displayed: the first 5 kernels are disorder detecting kernels with a high contrast and low average weight (ZHC), the sixth kernel is an order detecting kernel with a negative average weight and low contrast (NLC). In (b), the activation $\mathbf{Y}^{1,4}$, resulting from the convolution of kernel 4 with the input, is shown (for each of the 20 input configurations shown in figure 4.10). Resulting in the appearance of positive values in disordered areas (type D_9). In (c), the activation $\mathbf{Y}^{1,6}$, resulting from the convolution of kernel 6 with the input, is shown. The negatively magnetized areas light up (type $O(-)_9$).

The evaluation of the activations of the 4^{th} and 6^{th} channel (figure 4.11), helps to understand the working of both types of kernels on the input Ising configurations. Apart from the dimensionality reduction, we can infer the following working of the kernels on the input layer:

1. zero average, high contrast kernels (ZHC), will result in disorder highlighting activations D. Because $\mu \approx 0$, the average value of the input is shifted to zero, resulting in low values for both type of ordered areas (either positively or negatively aligned). Due to the high Δ value, the contrasts in the disordered regions are highlighted. Subsequently, the ReLU activation function ensures

[5] Later on, some notes will be made on the decision of these boundaries.

that the negative contributions are removed, yielding an average positive value in the disordered areas. These kernels can be referred to as **disorder detecting** kernels.

2. negative average, low contrast kernels (NLC), will result in activations of type $O(-)$ with high values in negatively magnetized areas. Because $\mu < 0$, the signs of the input are switched on average, resulting in positive values for negatively magnetized areas and negative values for positively magnetized areas, which are set to zero by the ReLU activation. Due to the low Δ value, the contrasts in the disordered regions are averaged out to zero as well. Resulting in an activation of type $O(-)$. (Perfectly analogous, PLC kernels will result in $O(+)$ activations.) These kernels can be referred to as **order detecting** kernels.

Similar results were obtained in [32], where a simpler convolutional network was investigated, consisting of a single convolutional layer with kernel size 2. The kernels were called either energy or magnetization detecting, respectively corresponding to the disorder and the order detecting kernels defined here.

It is true that the disorder and the order detecting kernels deduce energy- and magnetization- like order parameters. However, they are certainly not the real energy and magnetization. Still, it is very plausible that these energy-and magnetization-like order parameters will, just like the real energy and magnetization, also exhibit large variations at transition. These variations in the parameters, based upon which the network makes its classification, can explain the peak in the cost at transition temperatures found in section 4.1.1.

Two remaining types of kernels have not yet been discussed:

3. negative average, high contrast kernels (NHC), will highlight both negatively ordered areas (since $\mu < 0$) and disordered areas because of high Δ. (Perfectly analogous reasoning for PHC)

Tab. 4.4: Overview of the kernels of the first convolutional layer.

input	label	kernel	μ	Δ	label	activations	label
$\mathbf{Y}^{0,1}$	$O(+)_1$	$\mathbf{W}^{1,1,1}$	-0.0085	0.8074	ZHC	$\mathbf{Y}^{1,1}$	D_9
		$\mathbf{W}^{1,2,1}$	0.0008	0.9269	ZHC	$\mathbf{Y}^{1,2}$	D_9
		$\mathbf{W}^{1,3,1}$	-0.0102	0.6565	ZHC	$\mathbf{Y}^{1,3}$	D_9
		$\mathbf{W}^{1,4,1}$	-0.0106	0.9769	ZHC	$\mathbf{Y}^{1,4}$	D_9
		$\mathbf{W}^{1,5,1}$	0.0070	1.0089	ZHC	$\mathbf{Y}^{1,5}$	D_9
		$\mathbf{W}^{1,6,1}$	-0.1992	0.4829	NLC	$\mathbf{Y}^{1,6}$	$O(-)_9$

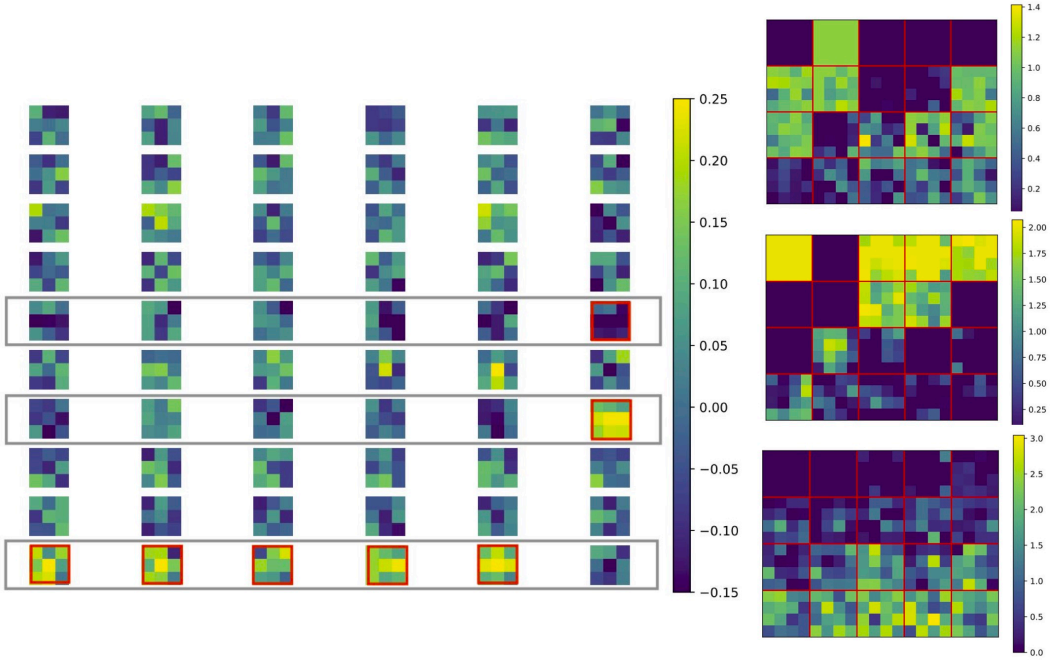


Fig. 4.12: On the left, the kernels of the second convolutional layer are displayed. On the right, the activations of the $5^{th}, 7^{th}$ and 10^{th} channel of the second convolutional layer are shown: $\mathbf{Y}^{2,5}$,which has high values in positively magnetized areas and disordered areas (type $(O(+)_{25} + D_{25})$), $\mathbf{Y}^{2,7}$,which has high values in negatively magnetized areas (type $O(-)_{25}$) and $\mathbf{Y}^{2,10}$, which lights up in areas of disorder (type $D(-)_{25}$). The kernels corresponding to these channels are indicated in grey. In red, the most determining kernels are indicated (summarized in table 4.5).

4. zero average, low contrast kernels (ZLC) have overall low weights and will therefore have limited contribution to the eventual output.

It should be noted that the sharp labeling of the kernels gives a slightly distorted view, moreover, it can be argued that the choice of boundaries μ^* and Δ^* is partly arbitrary. For example, the sixth kernel, $\mathbf{W}^{1,6,1}$, was labeled to have low contrast, but in fact the contrast is still considerable (table 4.4), resulting in an activation of mixed type $O(-) + fD$, lighting up strongly in the negatively magnetized areas but also partially, indicated by the fraction f, in the disordered areas (figure 4.11).

Second convolutional layer

The second convolutional layer makes the interpretation of the model more difficult. The concept of order detecting (NLC, PLC) and disorder detecting (ZHC) kernels is no longer applicable because the input activations are no longer Ising configurations ($O(+)_1$) and can belong to various types. However, the manner of interpreting is similar.

Figure 4.12 shows the activations of the fifth, the seventh and the tenth channel of the second convolutional layer. These channels were chosen because they have the largest contribution to the final output[6].

Using the same boundaries $\mu^* = 0.08$ and $\Delta^* = 0.5$ as before[7], the majority of kernels have $|\mu| < \mu^*$ and $\Delta < \Delta^*$ (type zero average, low contrast ZLC), which have only limited contribution to the eventual output. They can be omitted in the interpretation of the general picture. The determining kernels of channel 5,7 and 10 are:

- For channel 5: the 6^{th} kernel $\mathbf{W}^{2,5,6}$ works in on the 6^{th} activation of the first convolutional layer with a negative average and low contrast (NLC); flipping the signs of the activation. $O(-)_9 \rightarrow (O(+)_{25} + D_{25})$

- For channel 7, the 6^{th} kernel $\mathbf{W}^{2,7,6}$ works in on the 6^{th} activation with a positive average and low contrast (PLC), retaining the signs. $O(-)_9 \rightarrow O(-)_{25}$.

- For channel 10, the first 5 kernels $\mathbf{W}^{2,10,1-5}$ work in on the first 5 activations with a positive average and low contrast (PLC), retaining the signs and smoothening out the variations in the disordered areas $D_9 \rightarrow D_{25}$.

The scale of evaluation increases from 9 to 25 spins. These results are summarized in table 4.5.

Output layer

To obtain the desired output O_{256}, the scale of evaluation needs to encompass the complete lattice. Furthermore, an output of type O, can be reached in 2 basic ways:

[6] This information was inferred through the evaluation of the weights of the fully connected output layer

[7] Again it should be noted that the sharp labeling of the kernels gives a slightly distorted view and the choice of boundaries μ^* and Δ^* can be questioned, however, these simplifications allows to draw conclusions on the major trends of the dataprocessing

Tab. 4.5: Overview of the most determining kernels of the second convolutional layer.

activations 1	label	kernel	label	activations 2	label
$\mathbf{Y}^{1,6}$	$O(-)_9$	$\mathbf{W}^{2,5,6}$	NLC	$\mathbf{Y}^{2,5}$	$O(+)_{25} + D_{25}$
$\mathbf{Y}^{1,6}$	$O(-)_9$	$\mathbf{W}^{2,7,6}$	PLC	$\mathbf{Y}^{2,7}$	$O(-)_{25}$
$\mathbf{Y}^{1,1-5}$	D_9	$\mathbf{W}^{2,10,1-5}$	PLC	$\mathbf{Y}^{2,10}$	D_{25}

- $O = O(+) + fO(-)$, where the factor f makes sure that both contributions are balanced.

- $O = -D$

or a combination of both. Examining the weights of the fully connected output layer, it is found that the output y is approximately determined as

$$y \approx 4\langle \mathbf{Y}^{2,5} \rangle + 2\langle \mathbf{Y}^{2,7} \rangle - 3\langle \mathbf{Y}^{2,10} \rangle, \tag{4.5}$$

corresponding to the balancing of $O(+)$ and $O(-)$ with an extra contribution of $-D$.

4.1.5 Convergence, overfitting and early stopping

Convergence and overfitting

In section 4.1.2, it was established that fully connected neural networks perform worse than convolutional neural networks because of overfitting. This was due to the fact that FCNN need more parameters than CNN for similar performance (table 4.2). In this section, the overfitting will be visualized by plotting the evolution of C_{test} and C_{train} during training.

In the introduction on machine learning (chapter 1.1), it was noted that a good model has the capacity to generalize: During training the cost C_{train} on the training set $\mathbf{X}_{train}$ is minimized. However to evaluate the true performance of the model, the cost C_{test} on the testing set $\mathbf{X}_{test}$ is used. **Overfitting** occurs when C_{train} becomes significantly lower then C_{test}, indicating that the model has captured noise patterns that are specific for $\mathbf{X}_{train}$ and have nothing to do with global trends of the data. An overfitted model performs badly on new data.

In figure 4.13 the evolution of the training and testing cost during training is shown for both a fully connected and a convolutional network. The phenomenon of overfitting is clearly visible in the training of the fully connected model:

1. $C_{train} \approx C_{test}$, fast decrease (epoch 0-30)

2. $C_{train} < C_{test}$, decrease slows down (epoch 30-45)

3. C_{test} is minimal, the model has reached its optimal performance (epoch 45)

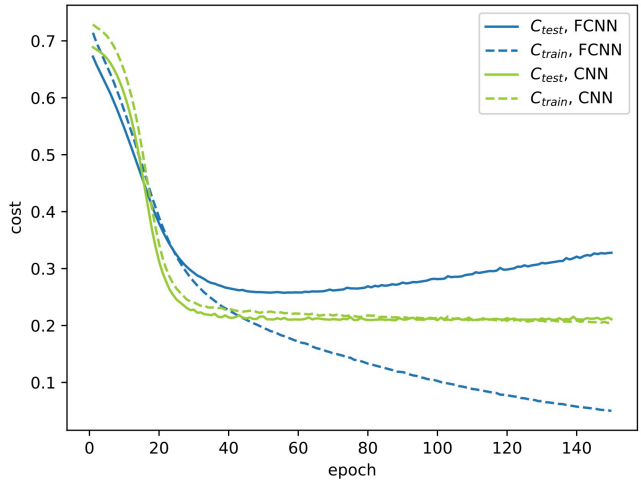


Fig. 4.13: Illustration of overfitting. Full line: loss on the testing set C_{test}. Dashed line: loss on the training set C_{train}. The blue graph shows the training of a fully connected network. The model reaches optimal performance at 45 epochs, afterwards overfitting occurs and C_{test} increases again. This is in contrast with the training of a convolutional network (green), which is not affected by overfitting. The performance stays approximately constant after convergence. (**Specifications of training:** modeltype: [CNNK3PP, Full], optimizer: SGD, batch size: 75, learning rate : 10^{-2}, # train (test) conf: 75(25))

4. $C_{train} < C_{test}$, C_{train} keeps decreasing at the expense of C_{test} which starts increasing again (epoch 45-150).

In contrast to the fully connected model, the convolutional model shows no signs of overfitting, corresponding to the results of section 4.1.2.

Early stopping

It becomes clear that the training of the models should be stopped as soon as they have reached their minimal testing cost C_{test}. To this end, the early stopping algorithm is introduced. This algorithm keeps track of the minimal cost C_{test} encountered during training. If the minimal cost was not updated for a certain amount of epochs (the patience), the model is considered to be converged and the training stops. The algorithm can be summarized as follows:

- set $C_{min} = \infty$, counter = 0
- choose patience, $\text{epoch}_{\text{max}}$
- for epoch $<$ $\text{epoch}_{\text{max}}$:
  - update the model
  - calculate C_{test}

- if $C_{test} < C_{min}$:
  - $C_{min} = C_{test}$
  - bestmodel = model
  - counter = 0
- else:
  - counter = counter + 1
  - if counter = patience:
    - model = bestmodel
    - break off training

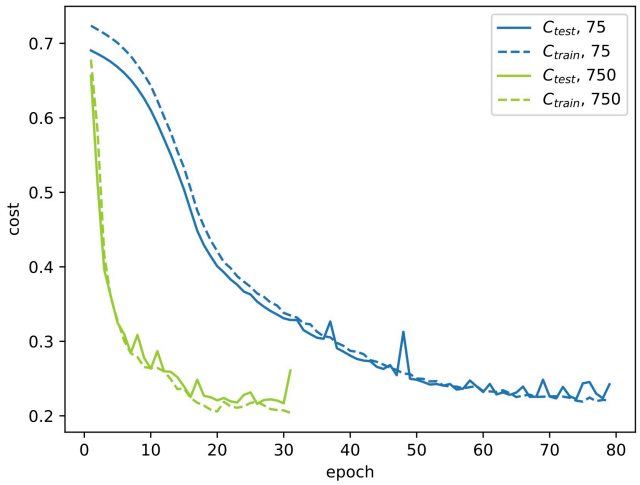


Fig. 4.14: Evolution of the losses during training with implemented early stopping algorithm. The patience was set to 5. Full line: loss on the testing set C_{test}. Dashed line: loss on the training set C_{train}. The blue and green curves represent two separate trainings respectively using 75 and 750 training samples. (**Specifications of training:** modeltype: CNNK3PP, optimizer: SGD, batch size: 64, learning rate : 10^{-2}, # train(test) conf: [75(25), 750(250)], patience: 5)

In figure 4.14 the evolution of the losses during training with implemented early stopping algorithm is shown. The algorithm meets the expectations:

- The training is completed before the maximum number of epochs is exceeded.
- The patience is sufficiently large; the algorithm is not interrupted prematurely because of a temporary fluctuations in C_{test}.

4.1.6 Weight initialization

[11] [33]

At the start of a training, the networks initial parameters (weights and biases) $\boldsymbol{\theta}_0$ are chosen, this process is referred to as weight initialization. In this section, the conditions on the weight initialization are discussed.

The Ising configurations consist of spin-values of +1 or -1, the average of every spin-value over the complete data set has approximately zero mean and standard deviation 1[8]. Because of this positioning around zero, it is reasonable to assume that the eventual parameters of the network will also be distributed around zero.

At first sight, it sounds sensible to initialize all parameters to zero since, on average, this is the closest we can bring them to their eventual value. In practise, this approach does not have the desired effect, it results in a very bad performing model. The reason for this is that weights, initialized to the same value and sharing the same connections, will become completely equivalent and will give rise to the same gradients, undergo the same updates and evolve in the exact same way. To illustrate, all weights in a fully connected layer will evolve together. Another example is the coupled evolution of kernels in a convolution layer. The network gets no chance to break the symmetry and therefore there is not enough freedom to increase performance; the performance stays constant with respect to the untrained model.

The parameters should be distributed around zero, but they cannot be exactly zero since this introduces unwanted symmetry in the model. The symmetry can be broken by initializing the weights to small random numbers.

The initial weights should be calibrated in order to prevent a layer by layer increase in the variance of the activations [11]. To illustrate this, consider the weighted sum of a neuron, $y = \mathbf{w}\,\mathbf{x}$, representing the basics of the way in which one layer influences the next (section 3.2). Let's estimate how variations on the input values var($\mathbf{x}$) will influence the variations of the output values var(y)[9]

$$\begin{aligned} \text{var}(y) &= \text{var}(\mathbf{w}\,\mathbf{x}) \\ &= \sum_{i=1}^{N} \text{var}(w_i x_i) \\ &= \sum_{i=1}^{N} \text{var}(w_i)\text{var}(x_i) \\ &= (N\text{var}(w))\,\text{var}(x) \end{aligned} \tag{4.6}$$

[8] data sets that do not have zero mean and standard deviation 1 often undergo a pre processing in which they are zero-centered and normalized. [11]

[9] Note that the bias is omitted in the weighted sum $y = \mathbf{w}\,\mathbf{x}$. Furthermore, to calculate the actual activation of a layer, the weighted sums should be evaluated by the non-linear activation function (section 3.2). However, to estimate the order of magnitude of the increasing variance, this simple weighted sum is a good enough approximation.

where N equals the number of input values included in the weighted sum. The first step is possible because w_i, w_j, for $i \neq j$, are linearly independent. The third step uses $\langle w_i \rangle = 0$ and $\langle x_i \rangle = 0$, the final step assumes that w_i, x_i are identically distributed for $i \in \{1, .., N\}$. Therefore, if we want to keep the variation constant from layer to layer, $\text{var}(y) = \text{var}(x)$, then

$$\text{var}(w) = \frac{1}{N}. \tag{4.7}$$

As said, N equals the number of input values included in the weighted sum. For a fully connected layer, N is simply equal to number of nodes of the previous layer. For a convolution layer with kernel size K, N equals the number of channels of the previous layer, D, times the number of weights per kernel: $D \times K \times K$.

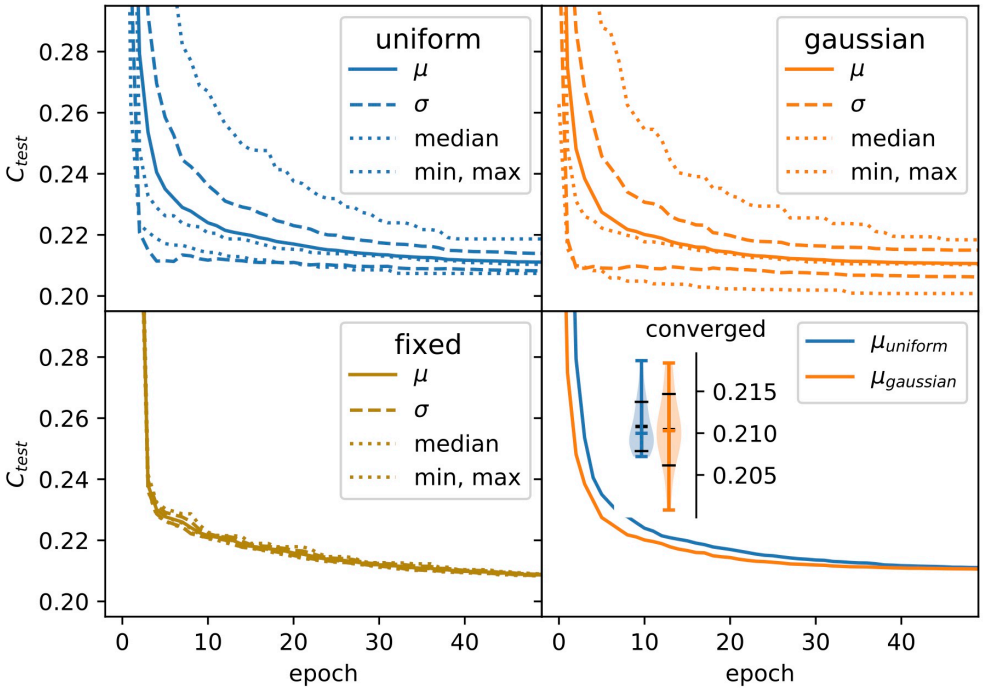


Fig. 4.15: Distribution of the loss evolution per epoch. Upper left: uniform initialization (25 trainings). Upper right: gaussian initialization (25 trainings). Lower left: fixed initialization (10 trainings). The lower right figure shows the average loss evolution for uniform and gaussian initialization. The inset contains the performance after convergence, where the marks indicate $[\mu - \sigma, \mu, \mu + \sigma]$. (**Specifications of training:** modeltype: CNNK3PP, optimizer: SGD, batch size: 64, learning rate : 10^{-2}, # train(test) conf: 750(250), patience: 5)

In figure 4.15, the distribution of the loss evolution per epoch is shown for 3 types of initialization:

1. The initial weights and biases are drawn from a uniform initialization distribution: $w_i, b \in [-\sigma, \sigma]$, where $\sigma = \frac{1}{\sqrt{N}}$. This is the default initialization procedure of PyTorch. A total of 25 trainings was used.

2. The initial weights are drawn from a gaussian distribution with standard deviation $\sigma = \frac{1}{\sqrt{N}}$. The biases are set zero. A total of 25 trainings was used.

3. One set of initial parameters $\boldsymbol{\theta}_0$ is drawn from the default distribution. These parameters are reused for all trainings. A total of 10 trainings was used.

For both the uniform and the gaussian initialization, the speed of convergence varies strongly for different initial parameters (upper left and right panel of figure 4.15). From the curves indicating the minimum, median and maximum performance per epoch, it becomes clear that most trainings converge faster then average, but they are countered by a few slow outliers. The cost-evolution is very similar for both types of initialization. However, some small differences were perceived (lower right graph of figure 4.15):

- The convergence speed seems to be slightly higher for gaussian initialization.

- On average, the eventual cost is slightly lower for gaussian initialization, but this difference is negligible compared to the standard deviation.

- The standard deviation on the performance is slightly higher for gaussian initialization.

The graph in the lower left corner, shows what happens if the same initial parameters are re-used for all 10 trainings. In this case, the transient varies much less, this is an extra confirmation that the speed of convergence depends strongly on the initial parameters. Future investigation, regarding the speed and stability of convergence, could entail the training of a large set of models using the default initialization and afterwards investigate the correlations between the performance and the initial parameters. In this way, one could probably find how the initialization distribution could be changed for better convergence.

Finally it should be noted that improving the speed and stability of convergence are often done by using a recently developed technique called Batch Normalization [16]. Batch Normalization boils down to the insertion of an extra function in every layer, which recenters and normalizes the activations. Apart from improving the speed and stability of convergence, Batch Normalization also reduces the importance of weight initialization, because the rescaling of activations makes layer-by-layer increasing variances impossible. The reason that Batch Normalization was not implemented in the used models is that it is known to counter the effect of L2-regularization [21]. Therefore, the choice was made to forego Batch Normalization and further investigate regularization (introduced in section: 3.7.3, implemented in section 4.1.7).

4.1.7 Regularization (weight decay)

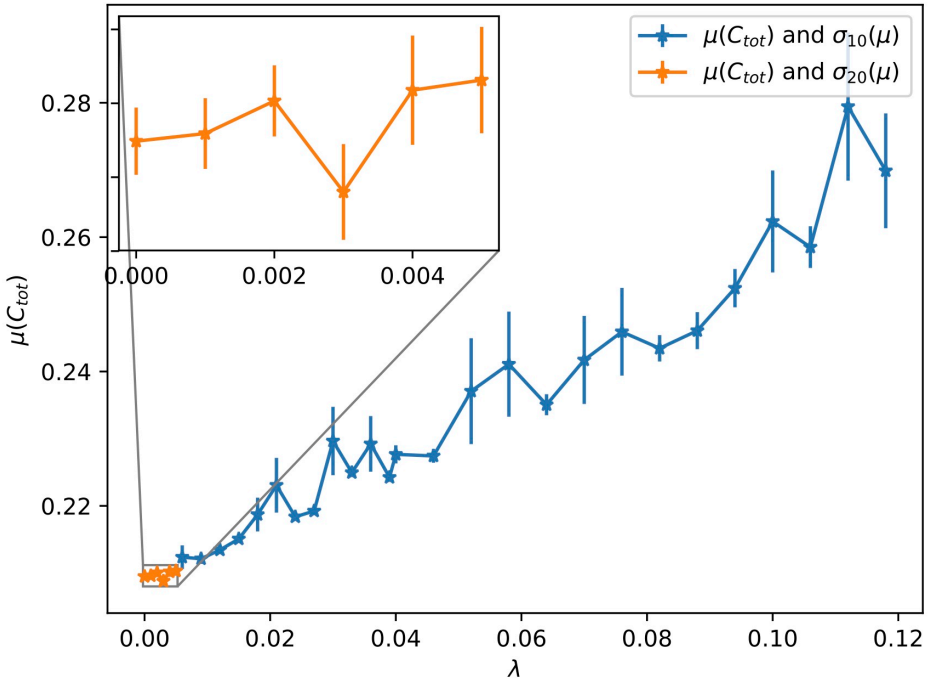


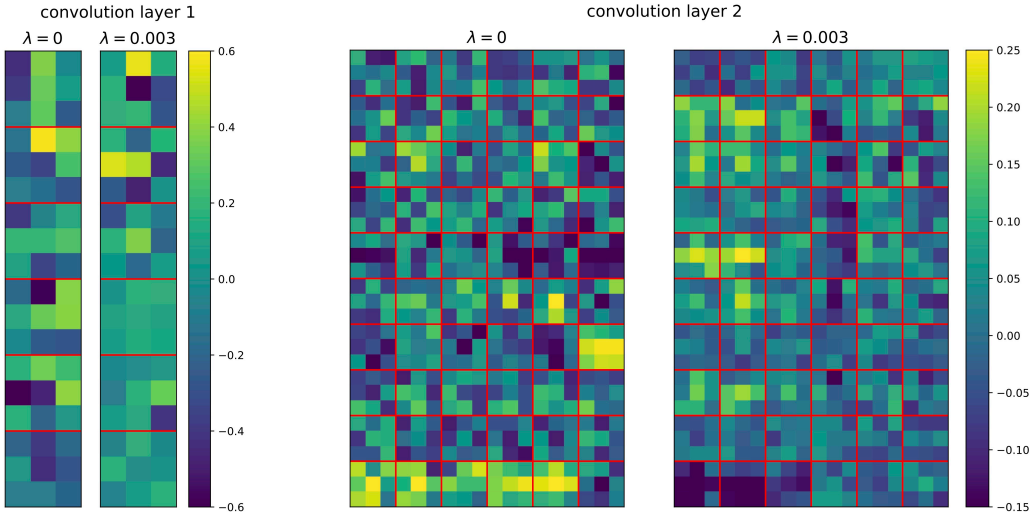
Fig. 4.16: For multiple values of the regularization hyperparameter λ, the average performance is calculated. Orange: average of 20 separate trainings. Blue: average of 10 separate trainings. *Note: σ is the estimated standard deviation on the average performance μ, not on the performance itself.* (**Specifications of training:** model: periodic padding with kernel size 3, optimizer: SGD, batch size: 20, learningrate: 10^-2, # train (test) conf: 750(250), patience: 5)

In section 3.7.3 of the previous chapter, a regularization term was added to the cost function in which large parameter-values are penalized. L2-regularization is described by the following expression:

$$C_{tot}(f(\mathbf{X}|\boldsymbol{\theta})) = C_{original}(f(\mathbf{X}|\boldsymbol{\theta})) + \lambda||\boldsymbol{\theta}||_2^2. \tag{4.8}$$

Excessive weights, often the result of noise on the training data, can cause large, misguided variance of the output, which decreases the generalizing capacity of the model.

In figure 4.16 the average performance is shown for multiple values for the regularization hyperparameter λ. The main conclusion is that their is no significant improvement of performance for any λ; for $\lambda \in [0, 0.005]$ the performance is approximately stable, for higher λ, the performance gets increasingly worse. Regularization might be a good approach for other models or problems but in this case, it offers no significant advantage.



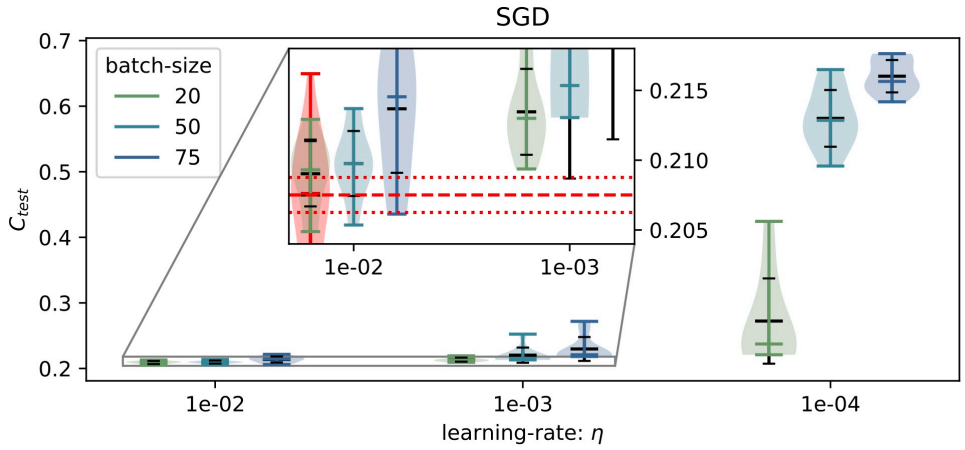
(a) (b)

Fig. 4.17: Kernels of the first (a) and second (b) convolution layer, for models trained with regularization hyperparameter $\lambda = 0$ and λ =0.003. The kernels that were obtained with regularization clearly tend to have lower weights in general. And the contrasts are located on one place in the kernel: e.g. one edge, one pixel is strongly weighted per kernel. For the unregularized kernels the weights are generally higher and there are multiple high weigths per kernel.

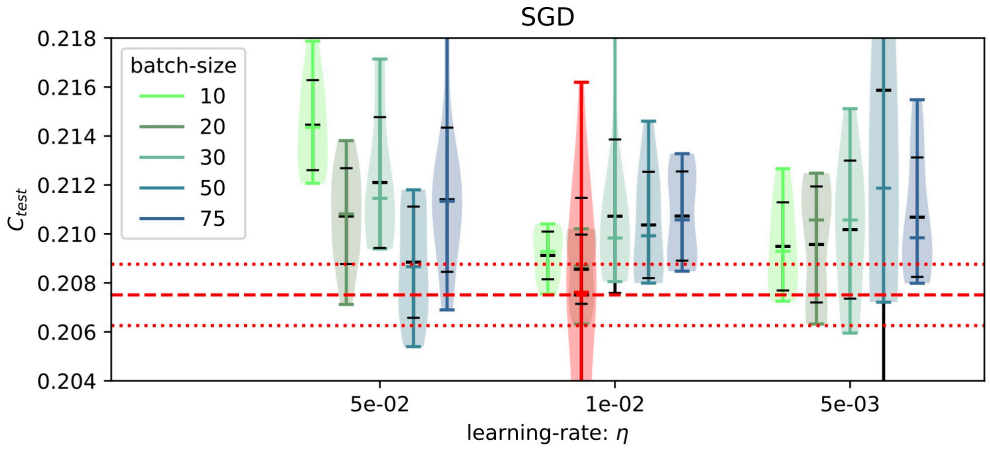
Figure 4.17 shows the kernels of the first (a) and second (b) convolution layer, for models trained with regularization hyperparameter $\lambda = 0$ and λ =0.003. The kernels that were obtained with regularization tend to have lower weights in general. Furthermore, the contrasts are located at one place in the kernel: e.g. one edge, one pixel, one connected group of pixels. For the unregularized kernels the weights are generally higher and apart from contrasts in edges and connected groups of pixels, checkered patterns are also frequent.

4.1.8 Comparison of optimization methods

In section 3.8.3, various optimization methods were discussed. In this section, the basic stochastic gradient descent (SGD) and a second moment method (Adam) will be compared. These methods depend on various hyperparameters: both Adam and SGD require a batch size and a learning rate η, Adam furthermore requires hyperparameters β_1 and β_2, which are the timescales for the first and second moment, and a regularization hyperparameter ϵ. To objectively compare both methods, these hyperparameters have to be optimized. A gridsearch algorithm is used to find for each method which set of hyperparameters minimizes the cost. Only the batch size and the learning rate are optimized, for β_1, β_2 and ϵ the default values are known to perform well [19].



(a) hyperparameters: $\eta \in [10^{-2}, 10^{-3}, 10^{-4}\ ] \times$ batch size $\in$ [20, 50, 75].



(b) hyperparameters: learning rate $\eta \in [5\ 10^{-2}, 10^{-2}, 5\ 10^{-3}\ ] \times$ batch size $\in$ [10, 20, 30, 50, 75].

Fig. 4.18: Performance of the model for optimization method SGD per set of hyperparameters. In (a), gridsearch is performed over large scale. In (b), the grid is narrowed around the best-performing hyperparameters found in (a). For every set of hyperparameters, the distribution of the results for 10 separate trainings is displayed. In red, the distribution of the results for 25 trainings with batch size 20 and learning rate 10^{-2} is indicated. Furthermore, the dashed red line gives an uncertainty interval for the estimation of $\mu_{10}(C_{test})$ (equation 4.9). (**Further specifications of training:** model type: ConvK3PP, # train(test) conf: 750(250), patience: 5)

In figure 4.18a, the performance of the model for optimization method SGD is shown for 9 sets of hyperparameters. The evaluated sets are : learning rate $\eta \in [10^{-2}, 10^{-3}, 10^{-4}\ ] \times$ batch size $\in$ [20, 50, 75]. The best performance is found for a learning rate 10^{-2} and a batch size 20. For the very small learning rate 10^{-4}, the cost seems to be extremely high, however, by examining the cost evolution of the training (figure 4.19), it becomes clear that the model was not yet converged within the imposed maximum number of epochs. Hence we can conclude that small learning rates will converge very slowly.

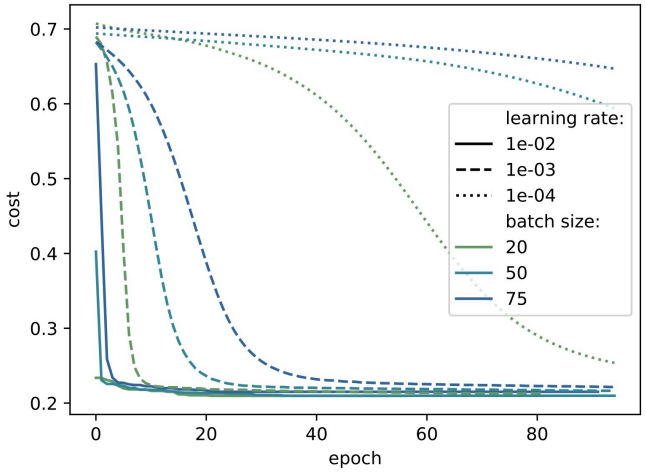


Fig. 4.19: Evolution of C_{test} for the hyper parameter sets, corresponding to figure 4.18a.

In figure 4.18b, the grid is narrowed around the best-performing hyperparameter-set: learning rate 10^{-2} and a batch size 20. In comparison with figure 4.18a the performances of all hyperparameter-sets are approximately the same. The best performance is again found at learning rate 10^{-2} and a batch size 20.

It is important to note however, that the best hyperparameter-set and trends in costs are difficult to establish, because the uncertainty is high at this scale. This is indicated by the largely overlapping errorbars. 10 trainings is not enough to be sure about the distribution.

To find out more about the order of uncertainty, the best performing hyperparameter-set was evaluated once more over 25 trainings (indicated in red on the figure). The results can be summarized:

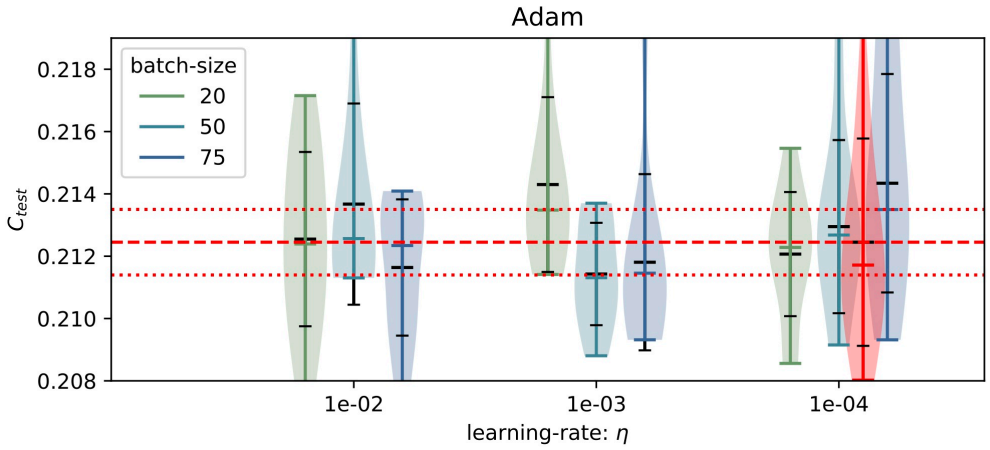
- $\sigma_{25} > \sigma_{10}$: The standard deviation was underestimated, but has the same order of magnitude.

- $\mu_{25} < \mu_{10}$: The average cost was slightly overestimated.

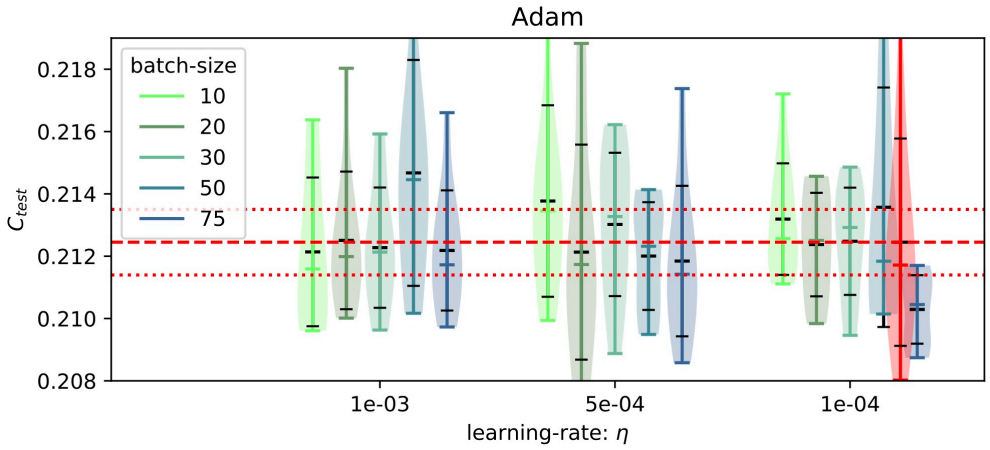
The law of large numbers dictates that:

$$\sigma(\mu_{10}(C_{test})) \approx \frac{\sigma_\infty(C_{test})}{\sqrt{10}} \approx \frac{\sigma_{25}(C_{test})}{\sqrt{10}}, \tag{4.9}$$

if we assume that $\sigma_{25}(C_{test})$ approximates the actual standard deviation of the distribution $\sigma_\infty(C_{test})$. The value $\sigma(\mu_{10})$ gives an uncertainty interval for the estimation of $\mu_{10}(C_{test})$ (dashed lines in the figure 4.18). In figure 4.18b, one can see that multiple hyperparameter-sets are close to the uncertainty interval of optimal performance.



(a) hyperparameters: $\eta \in [10^{-2}, 10^{-3}, 10^{-4}\ ] \times$ batch size $\in$ [20, 50, 75].



(b) hyperparameters: learning rate $\eta \in [1\ 10^{-3}, 5\ 10^{-4}, 1\ 10^{-4}\ ] \times$ batch size $\in$ [10, 20, 30, 50, 75].

Fig. 4.20: Performance for Adam per set of hyperparameters. In (a), gridsearch is performed over large scale. In (b), the grid is narrowed around the best-performing hyperparameters found in (a). For every set of hyperparameters, the distribution of the results for 10 separate trainings is displayed. In red, the distribution of the results for 25 trainings with batch size 64 and learning rate 10^{-4} is indicated. Furthermore, the dashed red line gives an uncertainty interval for the estimation of $\mu_{10}(C_{test})$ (equation 4.9). (**Further specifications of training:** modeltype: CNNK3PP, # train(test) conf: 750(250), patience: 5)

The same analysis was performed for Adam (figure 4.20). It is immediately clear that, in contrast to SGD, Adam is not very sensitive on the choice of hyperparameters.

Once more, the uncertainty on $\mu_{10}(C_{test})$ was established based upon the distribution of the performances for 25 trainings (this time with optimization method Adam, batch size 64 and learning rate 10^{-4}). Almost all sets of hyperparameters fall inside the indicated uncertainty range.

It is immediately clear that, in contrast to SGD, Adam is not very sensitive on the choice of hyperparameters. The result that the performance of Adam is less depen-

dent on the choice of hyperparameters than SGD, can be partially understood by considering the algorithms update rules (section 3.8.3):

SGD:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \eta \langle \nabla_{\boldsymbol{\theta}} C(\mathbf{x}, \boldsymbol{\theta}_t) \rangle_{\mathbf{x} \in \mathbf{B}}$$

Adam:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{\eta}_t \hat{\mathbf{m}}_t,$$

where the multiplication $\boldsymbol{\eta}_t \hat{\mathbf{m}}_t$ has to be seen element-wise with

$$\boldsymbol{\eta}_t = \frac{\eta}{\sqrt{\hat{\mathbf{s}}_t} + \epsilon} = \frac{\eta}{\sqrt{\mathbf{var}_t^2 + \hat{\mathbf{m}}_t^2} + \epsilon}.$$

$\hat{\mathbf{m}}$ and $\hat{\mathbf{s}}$ are respectively the first and second moment of the gradient $\langle \nabla_{\boldsymbol{\theta}} C(\mathbf{x}, \boldsymbol{\theta}_t) \rangle_{\mathbf{x} \in \mathbf{B}}$.

For SGD the learning rate η, determines the step size by which the parameters are updated, therefore, small learning rates will result in very large convergence times (figure 4.19). In comparison to SGD, Adam's optimization method is more self-regulating, adapting its effective learning rate $\boldsymbol{\eta}_t$, which is a function of the actual learning rate η, to the curvature of the cost function (section 3.8.3). This self-regulation translates in a lower sensitivity to the hyperparameter-values.

On the other hand, the optimal performance of SGD is better than the optimal performance of Adam.

4.2 Regression

In the previous section, neural networks were applied to perform a binary classification on Ising configurations, allocating them to either the ferromagnetic or the paramagnetic phase. In this section, regression problems are considered; we will investigate the ability of neural networks to predict the energy and magnetization of Ising configurations.

In contrast with classification, the model's output, y_{pred}, represents the prediction of a continuous feature, y, not the probability to belong to a certain phase (section 3.7.2). Therefore, a sigmoid activation function in the final layer is no longer appropriate; they were removed from the classification models used in the previous section.

The simplest cost function for a regression problem is the mean squared error between the model's prediction and the true value of the feature (section 3.7.2):

$$C_{MSE}(\mathbf{x}, \boldsymbol{\theta}) = (y - y_{pred})^2. \tag{4.10}$$

4.2.1 Energy

data sets

Once again, the same data set of Ising configurations, introduced in chapter 1.2.1, is used. This data set is an assembly of 20 data sets, which, through the application of the Metropolis Monte Carlo algorithm, are each sampled from the canonical ensemble of the Ising model at 20 corresponding temperatures surrounding the critical temperature: $T/T_c \in [0.5, ..., 1.5]$.

The total amount of example configurations used during training is varied to be either 15 000 or 150 000, such that the influence of the training set size can be estimated. The test set is the same for every training and consists of a total of 50 000 configurations.

For every configuration in the data set, the energy value is calculated by the analytic hamiltonian of the Ising model (for zero magnetic field):

$$E = -J \sum_{i,j=nn(i)}^{N} s_i s_j, \tag{4.11}$$

where N equals the number of spins in the lattice ($N = L^2, L = 16$). In fact, the parameter predicted by the neural network, is not the total energy, but the energy per degree of freedom: E/N.

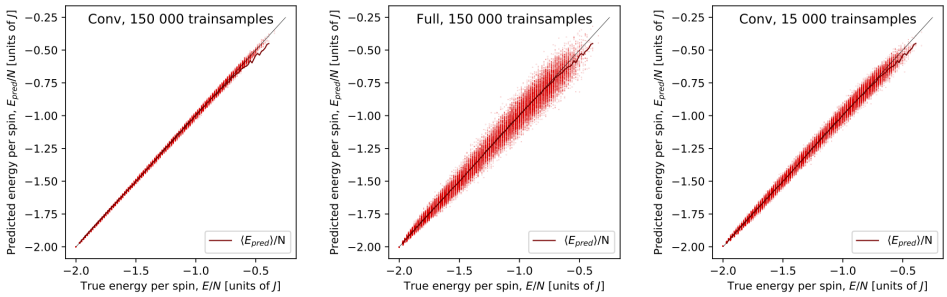
It should be noted that, even though regression is used, in reality, the energy of a finite Ising system is quantized:

$$\frac{E}{N} \in \left[-2J, \quad -2J + \tfrac{2\Delta E}{N}, \quad -2J + \tfrac{3\Delta E}{N}, \quad \ldots, \quad 2J - \tfrac{3\Delta E}{N}, \quad 2J - \tfrac{2\Delta E}{N}, \quad 2J\right] \tag{4.12}$$

where the cost to jump to an adjacent energy level equals $\Delta E = 4J$, except for the cost to break complete order (parallel or anti-parallel alignment), which requires $2\Delta E = 8J$. For increasing N, the scale of quantization $4J/N$ becomes very small, indicating that regression can be used [24].

The energy distributions of the data sets are plotted in histograms (where every bin corresponds to one energy level, $\Delta E = 4J$) in figure 4.22. Except for the very low energies, which are overly occupied because the majority of low temperature configurations corresponds to the completely ordered state, the energies are fairly evenly distributed. Furthermore, for $E/N > -0.5J$, the distribution falls to zero, because these higher energies are exceptional for temperatures $T/T_c < 1.5$.

Performance



(a) (b) (c)

Fig. 4.21: The predicted energy per spin E_{pred}/N for every Ising configuration ($L = 16$) in the testing set, is plotted as a function of the true energy per spin E/N. This was done for three separately trained models: a convolutional model trained with 150 000 training samples (a), a fully connected model trained with 150 000 training samples (b) and a convolutional model trained with 15 000 training samples (c). The dark red line indicates the average predicted energy $\langle E_{pred}\rangle_E/N$ as a function of the true energy. [24] (**Specifications of trainings:** model: [ConvK3PP, Full, ConvK3PP], optimizer: SGD, batch size: 64, learning rate: 10^{-2}, early stopping with maximum number of epochs: 500 and patience: 5)

Figures 4.21 and 4.22, display the results of three separately trained models:

- A convolutional neural network trained with 150 000 training samples.
- A fully connected neural network trained with 150 000 training samples.
- A convolutional neural network trained with 15 000 training samples.

Tab. 4.6: Training times of three distinct energy predicting models.

model type	# training samples	# epochs	time/epoch [s]	total time [s]
Conv	150 000	160	6.66	1065
Full	150 000	374	6.36	2378
Conv	15 000	152	0.91	139

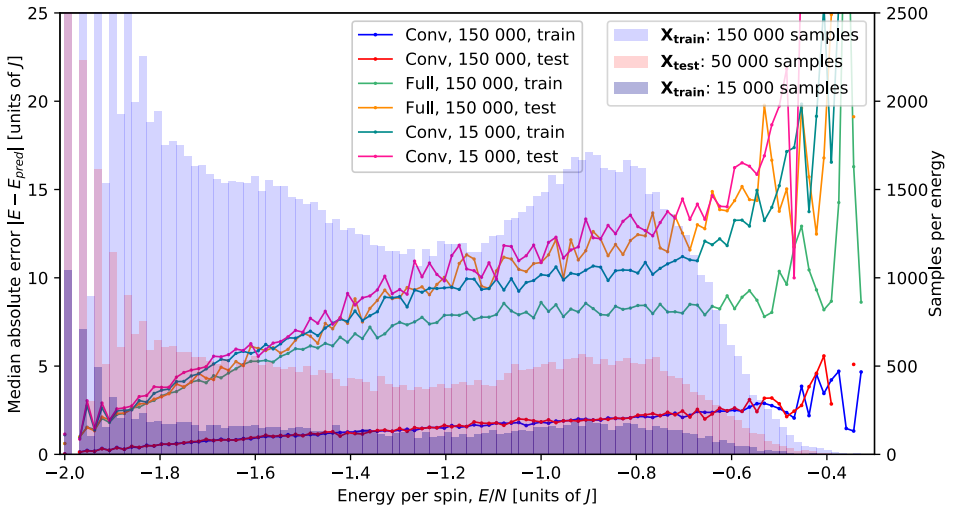


Fig. 4.22: The median absolute error of the predicted energy, $|E - E_{pred}|$ (left axis), is displayed as a function of energy per spin E/N for the three separately trained models. For each model, the results for both the test (shades of red) and training (shades of blue) configurations are shown. Furthermore the histograms of the three used data sets are shown (right axis): the large training set (blue), the testing set (red) and the small training set (blue). The configurations have linear size $L = 16$. [24]

In figure 4.21, the predicted energy per spin E_{pred}/N for every configuration in the testing set $\mathbf{X}_{test}$, is plotted as a function of the true energy per spin E/N (equation 4.11). Perfect predictions lie on the diagonal (the distance to the diagonal indicates the cost: $y - y_{pred} = \sqrt{C_{MSE}}$).

From the distribution of predictions we can therefore conclude that the convolutional network, trained with the large training set of 150 000 examples, performs best, followed by the same convolutional network trained with the small training set of 15 000 examples. The results of the fully connected network are dispersed the most, despite the use of the large training set of 150 000 examples.

Figure 4.22 displays the median absolute error of the predicted energy, median($|E - E_{pred}|$), as a function of energy per spin E/N, for all three trainings and, for every training, the results for both the test (red) and training (blue) configurations are shown.

For all three trainings, the error increases with increasing energy. When the energy increases ($E \to 0$), the configurations become more disordered; there are an increasing amount of domains and domain edges. Moreover, increasing degeneracy causes considerable variation. These tendencies make the calculation of the energy more challenging. Better performance could be reached by using a data set that contains a

higher amount of the disordered configurations. At minimal energy ($E/N = -2J$), the individual configurations are repeated many times since there are only two possible states (all spins aligned, either up or down). Note that, despite the limited amount of ordered states at minimal energy, for larger energies, the training set will contain only a miniscule fraction of the entire configuration space (since it consists of 2^{16^2} different configurations). Therefore, we are certain that the network has generalizing capacity; making good predictions for configurations it has never encountered before.

The convolutional network, trained with 150 000 training samples, has clearly superior performance. The median absolute error is for most energies of the order $2J$, which is smaller than the difference between adjacent Ising energy levels $\Delta E = 4J$. Therefore, we can conclude that for at least more then half of the Ising configurations, rounding off the predictions of the network to the nearest Ising energy level, will give rise to the exact energy of the configuration.

The median absolute error for the other two trainings is of the order $10J > \Delta E = 4J$; rounding off to the nearest Ising energy level, would lead to a median misclassification of approximately 2 energy levels.

Comparing the performance of the models on the training set with the performance the testing set, shows

- that there is no difference in performance for the convolutional model trained with 150 000 examples.

- that overfitting[10] occurs for both the convolutional model trained with 15 000 examples and the fully connected model trained with 150 000 examples. The overfitting is more outspoken for the fully connected model.

The fluctuations of the curves can be related to the finite number of configurations in each of the energy bins.

For energies exceeding $0.5J$, the predictions are bad and moreover, they are subject to large fluctuations. The network was insufficiently trained at these energies because they are underrepresented in the training set.

[10] When a model performs better on the configurations used during training than on new configuration that it never encountered before, the model is said to be overfitted. (chapter 1.1).

Recovering the phase transition

Data sets, generated by the Metropolis Monte Carlo algorithm, approximate the canonical ensemble and can therefore be used to estimate properties of many particle systems in thermal equilibrium [12, 36, 24].

The calculation of the average energy, $\langle E \rangle$, and the heat capacity

$$C = \frac{\partial \langle E \rangle}{\partial T} = \frac{\langle E^2 \rangle - \langle E \rangle^2}{kT^2}, \tag{4.13}$$

where k represents the Boltzmann constant, depends on the calculation of the energies corresponding to each configuration in the data set. For the Ising system, one can calculate the exact energies using the analytic hamiltonian (equation 4.11). The usage of regression networks, provides an alternative way to calculate the energy.

The average energy per spin, $\langle E \rangle / N$, and the specific heat capacity per spin C/N were calculated for each of the 20 temperatures included in the data set (the testing set: containing $50\,000 = 20 \cdot 2500$ configurations). The calculations were done for both the exact analytic hamiltonian and for each of the three energy predicting models. The results are presented in figure 4.23.

Both convolutional models predict the energy and heat capacity very well. The larger errors, specific to the model that was trained with a smaller training set, are averaged out in the evaluation of the statistical values $\langle E \rangle$ and C. The fully connected network shows visible deviations, but the results are still acceptable.

The phase transition and the critical temperature can be recovered from the fast change in energy and the related peak in the heat capacity. All three energy predicting neural networks, are capable to recover the phase transition through calculation of $\langle E \rangle$ and C.

4.2.2 Magnetization

For the prediction of magnetization, exactly the same analysis is possible. This subsection contains a short discussion of the most important results.

For every configuration in the data set, the exact magnetization is calculated:

$$M = \sum_{i=1}^{N} s_i. \tag{4.14}$$

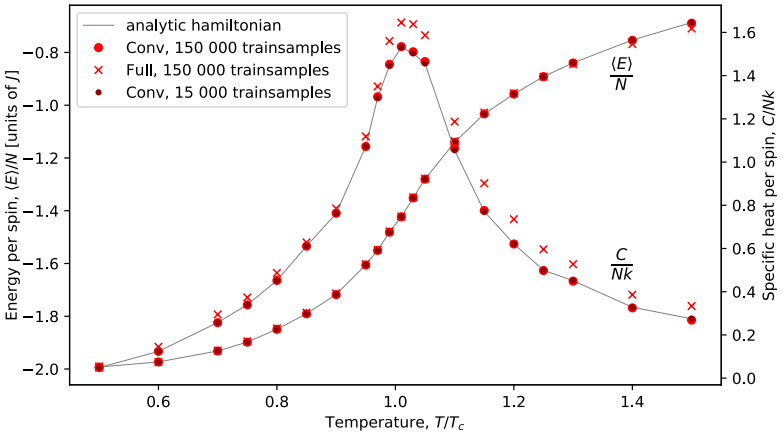


Fig. 4.23: For $L = 16$, the average energy per spin, $\langle E \rangle / N$, and the specific heat capacity per spin C/Nk were calculated using a testing set containing 2500 configurations per temperature. The calculations were done for both the exact analytic hamiltonian (grey line) and for each of the three energy predicting models (red marks). [24]

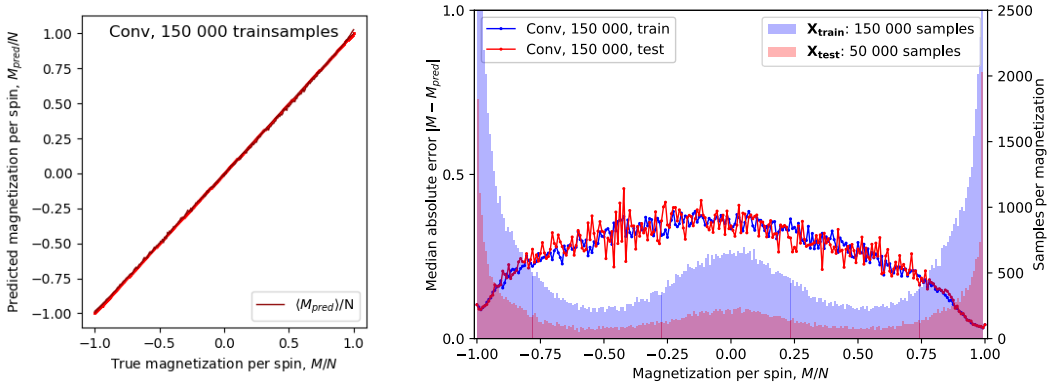
Analogous to the energy, the magnetization M of a finite Ising system is quantized. M/N ranges from -1 to 1 with a quantization step $\Delta M/N = 2/N$. For increasing N, the scale of quantization becomes very small, indicating that regression can be used.

In figure 4.24b, the magnetization distributions of the data sets are plotted in histograms (where every bin corresponds to one magnetization level ($\Delta M = 2$). The distribution peaks at -1 and 1 because the majority of ferromagnetic configurations correspond to the completely ordered states (all aligned, either downwards or upwards). At zero magnetization, a broader, lower peak represents the disordered, paramagnetic configurations. The intermediate magnetizations correspond to temperatures close to transition.

Figure 4.24 displays the performance of a convolutional neural network trained to predict the magnetization per spin M/N. The training set contains 150 000 training samples.

In figure 4.24a, the predicted magnetization per spin M_{pred}/N for every configuration in the testing set $\mathbf{X}_{test}$, is plotted as a function of the true magnetization per spin M/N. The resulting distribution lies very close to the diagonal, indicating good performance.

Figure 4.24b displays the median absolute error of the predicted magnetization, median($|M - M_{pred}|$), as a function of magnetization per spin M/N for both the test (red) and training (blue) sets. The error increases with decreasing absolute



(a) (b)

Fig. 4.24: The performance of a magnetization predicting convolutional network, trained with 150 000 examples of Ising configurations ($L = 16$), is displayed. Figure (a) shows the predicted magnetization per spin M_{pred}/N for every configuration in the testing set (containing 50 000 configurations), plotted as a function of the true magnetization per spin M/N. The average predicted magnetization $\langle M_{pred}\rangle_M/N$ as a function of true magnetization is displayed with a dark red line (but this line is almost indistinguishable). In figure (b), the median absolute error of the predicted magnetization, $|M - M_{pred}|$ (left axis), is displayed as a function of magnetization per spin M/N. The results for both the test (red) and training (blue) configurations are shown. Furthermore, the histograms of the data sets are displayed (right axis): the training set (blue) and the testing set (red). (**Specifications of trainings:** model: ConvK3PP, optimizer: SGD, batch size: 64, learning rate: 10^{-2}, early stopping with maximum number of epochs: 500 and patience: 5)

magnetization $|M|$. The explanation is the same as for the energy, when the magnetization approaches zero, the configurations become more disordered; there are an increasing amount of domains and increasing degeneracy causes considerable variation, making the calculation of the magnetization more challenging.

The median absolute error is lower then 0.5 for all magnetizations, which is smaller than the difference between adjacent Ising magnetization levels $\Delta M = 2$. Therefore, we can safely assume that with a few exceptions, for all Ising configurations, rounding off the predictions of the network to the nearest Ising magnetization level, will give rise to the exact magnetization of the configuration.

Analogous to the estimation of the average energy $\langle E\rangle$ and heat capacity C using the energy predicting networks, magnetization predicting networks can be used to estimate the average absolute magnetization $\langle|M|\rangle$ and susceptibility

$$\chi = \frac{\langle M^2\rangle - \langle|M|\rangle^2}{kT}, \tag{4.15}$$

from which the phase transition can be recovered.

5 Conclusions

In this work, it is illustrated that principal component analysis is capable of finding linear order indicating parameters, that allow to locate the phase transition. There is only one dominant principal component, which explains half of the variation. All other principal components have far lower variations, indicating that other non linear unsupervised learning techniques are more adequate to describe the second half of the variance (In reference [8] stochastic neighbor embedding (t-SNE) was used to present a non linear two-dimensional visualization of the Ising configurations.)

For classification neural networks, it is shown that the network calculates order and disorder indicating features based upon which the classification is made. Similar to the classical order and disorder indicating features of the Ising model (magnetization and energy) these features vary strongly at the phase transition, resulting in a characteristic error peak. Based on these considerations, it is suspected that there exists a lower limit to the cost function which is dependent on the physics of the model. In reference [7], it is shown that this error peak can be used for finite size scaling to correctly predict critical exponents of the infinite Ising model.

Convolutional regression neural networks can be trained to accurately predict energy and magnetization. The prediction of highly disordered states is challenging, but in contrast to classification, a theoretical lower limit on the error was not found. The ability of CNNs to learn operators has been shown to be applicable to various hamiltonians [24], both of short (such as variations of Ising model, e.g. the Potts model) and long range interactions (e.g. the Coulomb hamiltonian), providing accurate results while, in many cases, speeding up calculations enormously [24, 23].

For both classification and regression, it is shown that convolutional neural networks, which implement restrictions in their architecture based on the structure of the Ising configurations, perform better and are less sensitive to overfitting than fully connected neural networks which contain no such restrictions.

For all three types of machine learning, the data set used to train the model, should be sufficiently large to prevent overfitting and to improve performance.

Bibliography

[1] *A Practical Introduction to Deep Learning with Caffe and Python // Adil Moujahid // Data Analytics and more* (cit. on pp. 21, 23).

[2] Md Zahangir Alom, Theus Aspiras, Tarek M. Taha, and Vijayan K. Asari. „Skin Cancer Segmentation and Classification with NABLA-N and Inception Recurrent Residual Convolutional Networks“. In: *arXiv:1904.11126 [cs, eess]* (Apr. 2019). arXiv: 1904.11126 (cit. on p. 1).

[3] *Artificial neural network*. en. Page Version ID: 899336633. May 2019 (cit. on p. 21).

[4] Navin Bondade. *Recurrent Neural Network Detail Guide With Example And Applications*. en-US. Feb. 2019 (cit. on p. 23).

[5] Jeff Byers. „The physics of data“. en. In: *Nature Physics* 13 (July 2017), pp. 718–719 (cit. on p. 1).

[6] Wei Cai. *ME334 Introduction to Statistical Mechanics [Lecture Notes]* (cit. on pp. 5, 9).

[7] Juan Carrasquilla and Roger G. Melko. „Machine learning phases of matter“. en. In: *Nature Physics* 13.5 (May 2017). arXiv: 1605.01735, pp. 431–434 (cit. on pp. 18, 79).

[8] Juan Carrasquilla and Roger G. Melko. „Machine learning phases of matter“. en. In: *Nature Physics* 13.5 (May 2017), pp. 431–434 (cit. on pp. 45, 79).

[9] Francois Chollet. *How convolutional neural networks see the world*. Jan. 2016 (cit. on p. 29).

[10] *comp.ai.neural-nets FAQ, Part 3 of 7: GeneralizationSection - What is weight decay?* (Cit. on p. 36).

[11] *CS231n Convolutional Neural Networks for Visual Recognition* (cit. on pp. 32, 62, 63).

[12] H. Gould and J. Tobochnik. *Statistical and Thermal Physics: With Computer Applications*. Princeton University Press, 2010 (cit. on pp. 5, 6, 9, 76).

[13] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. 2nd ed. Springer, 2009 (cit. on pp. 1, 26).

[14] Kurt Hornik. „Approximation Capabilities of Muitilayer Feedforward Networks“. en. In: (), p. 7 (cit. on p. 25).

[15]Wenjian Hu, Rajiv R. P. Singh, and Richard T. Scalettar. „Discovering Phases, Phase Transitions and Crossovers through Unsupervised Machine Learning: A critical examination“. In: *Physical Review E* 95.6 (June 2017). arXiv: 1704.00080, p. 062122 (cit. on pp. 13, 15, 17).

[16]Sergey Ioffe and Christian Szegedy. „Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift“. In: *arXiv:1502.03167 [cs]* (Feb. 2015). arXiv: 1502.03167 (cit. on p. 65).

[17]jwymbs23. *Eye-sing Model*. en. Jan. 2018 (cit. on pp. 6, 7).

[18]Andre Peter Kelm, Vijesh Soorya Rao, and Udo Zolzer. „Object Contour and Edge Detection with RefineContourNet“. In: *arXiv:1904.13353 [cs]* (Apr. 2019). arXiv: 1904.13353 (cit. on p. 1).

[19]Diederik P. Kingma and Jimmy Ba. „Adam: A Method for Stochastic Optimization“. In: *arXiv:1412.6980 [cs]* (Dec. 2014). arXiv: 1412.6980 (cit. on p. 67).

[20]Jacques Kotze. „Introduction to Monte Carlo methods for an Ising Model of a Ferromagnet“. In: *arXiv:0803.0217 [cond-mat]* (Mar. 2008). arXiv: 0803.0217 (cit. on p. 6).

[21]Twan van Laarhoven. „L2 Regularization versus Batch and Weight Normalization“. In: *arXiv:1706.05350 [cs, stat]* (June 2017). arXiv: 1706.05350 (cit. on p. 65).

[22]Pankaj Mehta, Marin Bukov, Ching-Hao Wang, et al. „A high-bias, low-variance introduction to Machine Learning for physicists“. en. In: *arXiv:1803.08823 [cond-mat, physics:physics, stat]* (Mar. 2018). arXiv: 1803.08823 (cit. on pp. 1, 13, 21–23, 29, 31, 36, 39, 40, 43).

[23]Kyle Mills, Kevin Ryczko, Iryna Luchak, et al. „Extensive deep neural networks for transferring small scale learning to large scale systems“. In: *Chemical Science* 10.15 (2019). arXiv: 1708.06686, pp. 4129–4140 (cit. on p. 79).

[24]Kyle Mills and Isaac Tamblyn. „Deep neural networks for direct, featureless learning through observation: the case of 2d spin models“. In: *Physical Review E* 97.3 (Mar. 2018). arXiv: 1706.09779, p. 032119 (cit. on pp. 9, 72–74, 76, 77, 79).

[25]Tom M. Mitchell. *Machine Learning*. en. McGraw-Hill series in computer science. New York: McGraw-Hill, 1997 (cit. on pp. 1, 21, 23).

[26]*Multilayer perceptron*. en. Page Version ID: 891333013. Apr. 2019 (cit. on p. 26).

[27]Daniel Philps, Tillman Weyde, Artur d'Avila Garcez, and Roy Batchelor. „Continual Learning Augmented Investment Decisions“. In: *arXiv:1812.02340 [cs, q-fin, stat]* (Dec. 2018). arXiv: 1812.02340 (cit. on p. 1).

[28]*principal-component-analysis.png (800×800)* (cit. on p. 12).

[29]Hendrik Purwins, Bo Li, Tuomas Virtanen, et al. „Deep Learning for Audio Signal Processing“. In: *IEEE Journal of Selected Topics in Signal Processing* (2019). arXiv: 1905.00078, pp. 1–1 (cit. on p. 1).

[30]Sam T. Roweis. „EM Algorithms for PCA and SPCA“. In: *Advances in Neural Information Processing Systems 10*. Ed. by M. I. Jordan, M. J. Kearns, and S. A. Solla. MIT Press, 1998, pp. 626–632 (cit. on p. 20).

[31]Sebastian Ruder. „An overview of gradient descent optimization algorithms“. In: *arXiv:1609.04747 [cs]* (Sept. 2016). arXiv: 1609.04747 (cit. on pp. 39, 40).

[32]Philippe Suchsland and Stefan Wessel. „Parameter diagnostics of phases and phase transition learning by neural networks“. In: *Physical Review B* 97.17 (May 2018). arXiv: 1802.09876, p. 174435 (cit. on p. 57).

[33]Ilya Sutskever, James Martens, George Dahl, and Geoffrey Hinton. „On the importance of initialization and momentum in deep learning“. en. In: (), p. 14 (cit. on p. 62).

[34]Hao Tan, Licheng Yu, and Mohit Bansal. „Learning to Navigate Unseen Environments: Back Translation with Environmental Dropout“. In: *arXiv:1904.04195 [cs]* (Apr. 2019). arXiv: 1904.04195 (cit. on p. 1).

[35]„The thing about data“. en. In: *Nature Physics* 13.8 (Aug. 2017), p. 717 (cit. on p. 1).

[36]Jos Thijssen. *Computational Physics*. 2nd ed. Cambridge University Press, 2007 (cit. on pp. 8, 9, 76).

[37]Martino Trevisan, Luca Vassio, Idilio Drago, et al. „Towards Understanding Political Interactions on Instagram“. In: *arXiv:1904.11719 [cs]* (Apr. 2019). arXiv: 1904.11719 (cit. on p. 1).

[38]*Universal approximation theorem*. en. Page Version ID: 893483498. Apr. 2019 (cit. on pp. 18, 19, 25).

[39]Tom Vieijra. „Machine learning approaches to strongly correlated spin systems“. en. In: (), p. 134 (cit. on pp. 3, 40).

[40]Lei Wang. „Discovering Phase Transitions with Unsupervised Learning“. In: *Physical Review B* 94.19 (Nov. 2016). arXiv: 1606.00318, p. 195105 (cit. on p. 17).