FACULTY OF SCIENCES

AUTOREGRESSIVE NEURAL NETWORKS FOR SIMULATIONS IN CLASSICAL STATISTICAL PHYSICS

AARON GEMMEL

PROMOTOR: PROF. DR. JAN RYCKEBUSCH
SUPERVISORS: DR. JANNES NYS, CORNEEL CASERT and TOM VIEIJRA

A dissertation submitted to Ghent University in partial fulfilment of the requirements for the degree of Master of Science in Physics and Astronomy

ACADEMIC YEAR: 2020 – 2021

GHENT UNIVERSITY

GHENT UNIVERSITY:
Theoretical Nuclear and Statistical Physics
Department of Physics and Astronomy

REVIEWERS:
Prof. Dr. Ir. Toon Verstraelen
Prof. Dr. Jan Ryckebusch
Dr. Jannes Nys
Corneel Casert
Tom Vieijra

LOCATION:
Ghent, Belgium

TIME FRAME:
September 2020 – June 2021

ABSTRACT

Many problems in physics remain challenging to solve because of the large number of dimensions and/or degrees of freedom. The state space of physical systems typically scales exponentially with the system size. For example, the Ising model with N spins has 2^N possible configurations.

Traditional Monte Carlo techniques have been the major reliable choice to solve high-dimensional problems in statistical physics. However, these can suffer from long auto-correlation times between consecutive samples. In addition, local algorithms can get trapped in metastable states and prevent full exploration of the state space. This can be especially problematic for systems with many metastable states, for example, spin glasses.

Recently, Machine Learning has been successful in solving some of these complex physical problems. In fact, generative Neural Networks can be used to efficiently generate uncorrelated configurations for physical systems. In Machine Learning the complexity in solving the problem is shifted to the model's training process. In this study, an autoregressive ansatz is used to learn the Boltzmann distribution of one of the most widely studied physical systems: the Ising model. We use a Recurrent Neural Network architecture to model the spin chains $\mathbf{s} = (s_1, \ldots, s_N)$ as input sequences and parameterise each conditional probability $q_\theta(s_i \mid \mathbf{s}_{<i})$ separately. The conditionals are combined into a normalised probability $q(\mathbf{s}; \theta)$ to approximate the Boltzmann distribution $p(\mathbf{s})$. If the network is successful at capturing the distribution, it can efficiently generate configurations of the system which in turn can be used to estimate physical quantities. As a result, Machine Learning is a powerful alternate and/or complementary method to traditional Monte Carlo simulations.

In chapter 1 and chapter 2 we introduce classical statistical physics and Machine Learning, in which the reasons (the why) and the methods (the how) of applying Machine Learning to the physical system, are discussed. We then train the network on data, generated with a Markov Chain Monte Carlo algorithm, and show that it can sample the correct configurations and describe the physics of the system by comparing it to Monte Carlo estimations (chapter 3). Third, the Recurrent Neural Network is trained on data generated by the network and for which the data gathering step is completely omitted (chapter 4). We show that this alternative training approach is equally capable of describing the distribution of the physical system and show the effect of symmetrising the variational ansatz and sampling procedure. The advantages and disadvantages of training with and without a data

set are compared to each other and discussed. Finally, the variational method is applied to one-dimensional spin systems with random and long-range interactions (chapter 5). We investigate if and how the network architecture has to be modified to describe the correlations in the system. In this chapter, we show the advantages of adapting the architecture to the physical system under study.

SAMENVATTING

Veel uitdagende problemen in de fysica zijn moeilijk op te lossen door het grote aantal dimensies en/of vrijheidsgraden. De toestandsruimte van fysische systemen schaalt typisch exponentieel met de grootte van het systeem. Bijvoorbeeld, het welgekende Ising model met N spins heeft 2^N mogelijke configuraties.

Traditionele Monte Carlo technieken zijn al tientallen jaren een betrouwbare keuze om moeilijke problemen in de statistische fysica op te lossen. Tijdens simulaties kunnen lange auto-correlatietijden tussen opeenvolgende configuraties problemen veroorzaken. Ook kunnen lokale update algoritmes vast komen te zitten in metastabiele toestanden, waardoor niet de volledige toestandsruimte wordt verkend. Dit kan vooral een probleem vormen bij systemen met meerdere metastabiele toestanden zoals "spin glasses".

Machinaal leren is in staat om complexe fysische problemen op te lossen. We kunnen generatieve neurale netwerken gebruiken om efficiënt ongecorreleerde configuraties van fysische systemen te simuleren. Hierbij wordt de complexiteit van het probleem verschoven naar een moeilijk optimalisatieproces. In deze studie wordt een autoregressief model gebruikt om de Boltzmann distributie van het Ising model te leren. Een terugkerend neural netwerk wordt gebruikt om spinconfiguraties $\mathbf{s} = (s_1, \ldots, s_N)$ te modelleren als input zodat elke conditionele waarschijnlijkheid $q_\theta(s_i \mid \mathbf{s}_{<i})$ apart wordt geparametriseerd. De conditionele waarschijnlijkheden worden gecombineerd tot een genormaliseerde waarschijnlijkheid $q(\mathbf{s}; \theta)$ die de Boltzmann distributie $p(\mathbf{s})$ benadert. Als het netwerk een goede beschrijving van de distributie geeft, kunnen er efficiënt nieuwe configuraties worden gesimuleerd. Deze worden dan gebruikt om fysische grootheden te schatten. Machinaal leren is dus een alternatieve en/of complementaire methode op traditionele Monte Carlo simulaties.

In hoofdstuk 1 en hoofdstuk 2 worden klassieke statistische fysica en machinaal leren geïntroduceerd, waarin we bespreken waarom en hoe machinaal leren toegepast wordt op fysische systemen. Dan wordt een neuraal netwerk getraind op een verzameling Boltzmann configuraties, gegenereerd met Markov Chain Monte Carlo, en tonen we aan dat het netwerk in staat is om de fysica van het systeem te beschrijven door te vergelijken met Monte Carlo simulaties (hoofdstuk 3). Vervolgens wordt het netwerk getraind op configuraties gegenereerd door het netwerk zelf (hoofdstuk 4). Hiervoor zijn er dus geen a priori Boltzmann configuraties nodig. We tonen aan dat deze alternatieve trainingsmethode ook in staat is om de distributie van het systeem te beschrijven. We onderzoeken hoe de sym-

metrieën van het systeem in de variationele ansatz worden geïmplementeerd en welke impact dat heeft op de resultaten. De voor -en nadelen van trainen met of zonder a priori data worden vergeleken en besproken. Tenslotte wordt de variationele methode toegepast op één-dimensionale spinsystemen met willekeurige en langedrachtinteracties (hoofdstuk 5). We onderzoeken hoe de architectuur moet worden aangepast om correlaties in het systeem te beschrijven. We tonen de voordelen van een architectuur die rekening houdt met de fysica van het systeem.

ACKNOWLEDGEMENTS

Deze thesis is mogelijk gemaakt door de vele ondersteuning die ik doorheen het jaar gekregen heb.

Ik zou eerst en vooral professor Ryckebusch willen bedanken om deze thesis voor te stellen en mij te verwelkomen in het team. Dank u wel om de tussenversies van feedback te voorzien zodat ik dit eindresultaat kon neerzetten.

Ook hartelijke dank aan mijn begeleiders Jannes, Corneel en Tom, om tijdens talloze online meetings met mij samen te zitten, te luisteren en te discussiëren. Bedankt om op mijn vele vragen en e-mails te antwoorden en mij uit de problemen te helpen waar nodig.

Dank u mama, om de volledige bundel na te lezen op spellings -en grammaticafouten. Het was waarschijnlijk niet gemakkelijk om jezelf doorheen de codetaal te sleuren, maar het heeft geholpen om er een mooi en leesbaar geheel van te maken. Papa, bedankt om mij wijze raad te geven in het uur van nood.

Ik wil graag mijn vriendjes bedanken die mij doorheen dit lastig jaar hebben gesleurd: Ilka, Stephan en Yarrik. Jullie motiveerden mij om elke dag naar de bib te komen en verder te werken aan de thesis. De gezellige middagen met pizza's en brownies waren een bonus. Fien, bedankt om mij te ondersteunen tijdens de moeilijke periodes dit jaar. En ook aan alle anderen die mij een duwtje hebben gegeven onderweg: superbedankt!

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
ERM	Empirical Risk Minimization
FFNN	Feedforward Neural Network
GRU	Gated Recurrent Unit
GPU	Graphics Processing Unit
KL	Kullback-Leibler
MC	Monte Carlo
MCMC	Markov Chain Monte Carlo
MCS	Monte Carlo step per spin
MSE	Mean Squared Error
MD	Molecular Dynamics
ML	Machine Learning
MLE	Maximum Likelihood Estimation
NIS	Neural Importance Sampling
NLL	Negative Log-Likelihood
NN	Neural Network
RNN	Recurrent Neural Network
RSB	Replica Symmetry Breaking
SGD	Stochastic Gradient Descent

α	Power-law exponent for the correlations in spin glasses
β	Inverse temperature of the system
$\beta^*(t)$	Annealed inverse temperature at step t
Cov	Covariance
η	Learning rate
$\langle O \rangle$	Ensemble average of a macroscopic quantity O
$\mathcal{D}$	Set of data points
$\mathcal{N}$	Normal distribution
$\mathcal{U}$	Uniform distribution
$\omega_{xx'}$	Trial step probability in an MCMC algorithm
$\pi(x)$	Initial probability distribution of a Markov chain
Σ	Fully connected layer
σ	Power-law exponent for long-range interactions
τ	Correlation time
θ	Set of parameters of a model
Var	Variance
$\mathbf{h}$	Hidden vector
$\mathbf{s}$	Spin chain configuration
$\mathbf{s}_R$	Reflection of the spin configuration $\mathbf{s}$ between the boundaries
$\widehat{O}$	Estimate of the quantity O
$\widetilde{w}$	Unnormalised weight for NIS
$A_{xx'}$	Acceptance probability in an MCMC algorithm
C	Normalisation constant of a distribution
$C(i,j)$	Disconnected correlation function
$C_4(i,j)$	Bond average of the squared disconnected correlation function

c_g	Gradient clipping value
d	Dimension of the system
d_u	Upper critical dimension of spin glasses
D_{KL}	Kullback-Leibler divergence
E	Energy of the system
F	Free energy of the system
F_q	Variational free energy of the distribution q
$G(i,j)$	Connected correlation function between spin s_i and s_j
H	External magnetic field
h_d	Hidden dimension
J	Interaction strength between spins, also called the coupling
L	Negative log-likelihood loss
l	Likelihood loss
m	Average magnetisation per spin
m_b	Batch size for simulating MCMC with the batch means method
N	Size of the system
n	Number of samples
N_{eq}	Number of equilibration steps for MCMC
N_{mc}	Number of MC steps for MCMC
N_A	Number of annealing steps
n_b	Batch size for training a ML model
N_F	Number of steps after annealing
n_l	Number of layers
N_T	Number of training steps
n_T	Number of training samples
$p(x)$	Probability distribution, often used for the Boltzmann distribution
P_θ	Number of parameters of a model
$q(x;\theta)$	Model probability distribution, often used for the RNN distribution

q_{sym}	Symmetrised model probability distribution
r_{ij}	Distance between spin s_i and s_j
S	Entropy of the system
s_i	Single spin of the system
T	Temperature of the system
$T(x \to x')$	Transition matrix of a Markov chain
T_c	Critical temperature of the system
w	Normalised weight for NIS
x	Random variable
x_0	Initial state of a Markov chain
Z	Partition function
z	Dynamical exponent
z_b	Average coordination number

Part I

INTRODUCTION

1 STATISTICAL PHYSICS

1.1 STATISTICAL MECHANICS

Many complex problems in physics are characterised by a large number of degrees of freedom. Typically, these systems have many different state configurations. This makes it difficult to study and treat each state of the system individually.

For example, a microscopic description of a gas in a container requires the positions and velocities of each gas particle. Since a macroscopic gas contains a lot of particles, think of the order 10^{23}, a microscopic study would be very expensive and is in practice impossible. However, often we are not interested in the behaviour of each particle of the gas, but rather in the behaviour of the system as a whole.

Statistical mechanics and thermodynamics describe such systems by looking at statistical averages of macroscopic quantities and probability distributions of the state configurations. In particular, a large fraction of the systems naturally approaches a certain equilibrium in which averages of macroscopic quantities become time-independent. For now, the focus is on equilibrium statistical mechanics.

If a system is in equilibrium with a heat bath, then it is described by a *canonical ensemble*. Suppose the system is once again a gas but placed in contact with a heat bath at temperature T. Each microscopic configuration of positions and velocities of the particles represents a possible state of the system x. This canonical ensemble is characterised by the *Boltzmann distribution*, which is the probability that the system is in the state x:

$$p(x) = \frac{1}{Z(\beta)} \exp\left(-\beta E(x)\right), \tag{1}$$

where $\beta = 1/kT$ is the inverse temperature of the system and $E(x)$ the total energy of the system in state x. The normalising constant is called the partition function $Z(\beta) = \sum_x \exp\left(-\beta E(x)\right)$, which is a sum or integral over all possible configurations of the system.

To study the general behaviour of the system, including fluctuations, correlations and responses of physical variables, some important quantities have to be calculated or estimated:

- The free energy of the system

$$F = -\frac{1}{\beta} \ln Z. \tag{2}$$

This requires the partition function Z which contains a sum or integral over all possible configurations of the system. From the free energy and the partition function, a lot of other important quantities can be derived.

- Averages of macroscopic quantities O, for example, the magnetisation in spin systems. In practice this is done by estimating ensemble averages

$$\langle O \rangle = \frac{1}{Z(\beta)} \sum_x O(x) \exp\left(-\beta E(x)\right). \tag{3}$$

When discussing macroscopic quantities, for example, the magnetisation M in a magnetic system, this always refers to the time-independent average magnetisation $\langle M \rangle$ from (3).

which is again a sum over all possible configurations of the system. Although the partition function shows up in (3), it is sometimes possible to calculate the averages directly without explicit knowledge of the partition function.

To study the most interesting properties of a many-body system, one has to perform a sum or integral over a large number of states. For example, a system with N spins where each spin can take on two discrete values has 2^N possible states. Hence for large systems, the sum is very difficult to deal with. In particular, it is often interesting to study the behaviour in the thermodynamic limit; then the sum even contains an infinite number of states. In any case, it is difficult to find an exact analytical solution for most systems and one has to resort to other methods [37]. Here we focus on using computational techniques to solve these highly complex and high-dimensional problems.

There are essentially two popular computational methods to study many-body systems:

- *Monte Carlo (MC)*: this technique introduces artificial dynamics using a certain degree of randomness to simulate the system so that configurations are sampled according to the Boltzmann distribution. Since this method introduces artificial dynamics, it is unsuited to study dynamical quantities such as the *velocity auto-correlation function (VAF)* [47].

- *Molecular Dynamics (MD)*: this technique solves the classical equations of motion directly and simulates the system in the function of time. This makes it possible to study the dynamical evolution of the system. Once the system has reached equilibrium, configurations sampled according to the Boltzmann distribution can be taken from the simulation [47].

Especially the Monte Carlo technique is easy to implement and can obtain very good results for a lot of applications in statistical mechanics. We are often only interested in the global static behaviour of the system and not necessarily in the dynamical evolution. This is especially the case for lattice models such as the famous *Ising model*.

1.2 THE ISING MODEL

One of the most important and simple models in statistical physics is the Ising model. Its simplicity and emergent features, such as phase transitions, make it an ideal candidate to study new methods and techniques.

The Ising model is a magnetic system of neighbouring spins $\{s_i\}_{i\geqslant 1}$: each spin either has spin-up ($s_\uparrow = +1$) or spin-down ($s_\downarrow = -1$). The spins interact with each other through a coupling constant J and with a magnetic field H. The Ising model is interesting because no equation describes the dynamical evolution of the system. Instead, the global behaviour of the system is given by the total energy or Hamiltonian [20]

$$E = -J \sum_{\langle i,j \rangle} s_i s_j - H \sum_i s_i. \tag{4}$$

If $J > 0$, the system is ferromagnetic and neighbouring spins try to align. For $J < 0$, neighbouring spins try to push each other into an opposite spin and is the system antiferromagnetic.

The first term represents the interaction between nearest neighbouring spins. The second term tries to align the spins to an external magnetic field H. While most interesting properties are also present in the absence of a magnetic field ($H = 0$), from a computational perspective the magnetic field is easily implemented. From now on, suppose that the system is ferromagnetic ($J > 0$) and can have any value for the magnetic field H.

Solving the system comes down to finding the partition function Z. Given the partition function, it is possible to calculate the relevant physical quantities using equations from statistical mechanics. Exact solutions have been found for the one -and two-dimensional case. Here, the focus is on the one-dimensional Ising model, which does not exhibit a phase transition for any finite temperature T. Using the *transfer matrix* method, the solution of the one-dimensional Ising model with N spins can be found as [20]

$$\begin{aligned} Z_N &= \lambda_+^N + \lambda_-^N \\ \text{with} \quad \lambda_\pm &= e^{\beta J}\cosh(\beta H) \pm \left(e^{-2\beta J} + e^{2\beta J}\sinh^2(\beta H)\right)^{1/2}. \end{aligned}$$

The free energy per spin $f = F/N$ can be calculated using (2), which in turn can be used to find the other relevant physical quantities, summarised in Table 1. These are the total energy of the system $E = \frac{\partial \beta F}{\partial \beta}$, the entropy $S = \beta^2 \frac{\partial F}{\partial \beta}$ and the connected correlation function $G(i,j) = \langle s_i s_j \rangle - \langle s_i \rangle \langle s_j \rangle$, which represents the correlations between spin s_i and s_j. For a translationally invariant system such as

PHYSICAL QUANTITY	EXPRESSION
Free energy F	$-\frac{1}{\beta} \ln Z$
Energy E	$-J \sum_i s_i s_{i+1} - H \sum_i s_i$
Entropy S	$\beta E + \ln Z$
Magnetisation M	$\sum_i s_i$
Correlations $G(r)$	$\langle s_0 s_r \rangle - m^2$

Table 1: *An overview of the physical properties that are considered for studying the one-dimensional Ising chain, each with the expression used to calculate it in the canonical ensemble. Since the parameters β, J and H only appear as βJ and βH in the Boltzmann distribution (1) with energy (6), we consider the quantities βE and βF instead.*

the one-dimensional Ising, the correlation function only depends on the distance between the spins $r := |i - j|$. In practice, the correlations are estimated by averaging over all spin pairs a distance r apart.

Since the system is magnetic, with up and down spins, it is also interesting to look at the total magnetisation of the system $M = -\frac{\partial F}{\partial H}$. In the thermodynamic limit $N \to \infty$, the solution can be found as [20]

$$m = \frac{M}{N} = \frac{\sinh(\beta H)}{\left(\sinh^2(\beta H) + e^{-4\beta J}\right)^{1/2}}. \tag{5}$$

The important emergent behaviour of magnetic systems such as the Ising model are phase transitions: the system can be in different phases or states depending on the parameters, in this case, the temperature β, coupling constant J and magnetic field H. A phase transition in such a magnetic system occurs between an "ordered" or ferromagnetic state, where spontaneous magnetisation is possible ($M \neq 0$ when $H = 0$) and a "disordered" or paramagnetic state, where no spontaneous magnetisation occurs ($M = 0$ whenever $H = 0$). From the solution (5), it can be seen that $m = 0$ whenever $H = 0$ for any finite temperature $T > 0$. Only for $T \to 0$ or $\beta \to \infty$, is spontaneous magnetisation possible. As mentioned before, the one-dimensional Ising model does not exhibit a phase transition for finite temperatures and is not well suited to study the behaviour of systems near phase transitions.

Further, it is important to note that for finite systems the solution depends on the chosen boundary conditions. This is very relevant for computational simulations where the system is finite due to computational constraints. The two most common boundary conditions are:

- *Open boundary conditions*: the first s_1 and last spin s_N are not connected to each other. The one-dimensional chain is open on

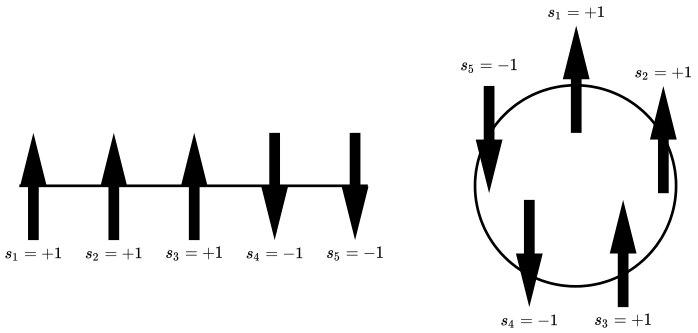

Figure 1: *An example of a one-dimensional Ising chain with* $N = 5$ *spins using open boundary conditions (left) and periodic boundary conditions (right).*

both sides and the first and last spin only have one interacting neighbour each. The energy (4) can be rewritten as

$$E = -J \sum_{i=1}^{N-1} s_i s_{i+1} - H \sum_{i=1}^{N} s_i. \tag{6}$$

- *Periodic boundary conditions*: the first s_1 and last spin s_N are connected to each other: $s_{N+1} \coloneqq s_1$. The one-dimensional chain essentially forms a ring where each spin interacts with two neighbouring spins. The energy (4) looks like

$$E = -J \sum_{i=1}^{N} s_i s_{i+1} - H \sum_{i=1}^{N} s_i. \tag{7}$$

Note that the number of terms in the first sums are different, $N-1$ and N respectively and that the solution is different in both cases for finite system sizes. Especially for small system sizes, the finite-size effect can cause a substantial deviation of the analytical solution for $N \to \infty$ depending on the boundary conditions. The choice of boundary conditions should not matter in the thermodynamic limit where the solution for both types converge to (5). A visualisation of the different boundary conditions can be seen in Figure 1.

1.3 SPIN GLASS MODEL

Spin glasses are spin systems characterised by magnetic interactions, randomness and frustration [35]. Contrary to the Ising model, spin glasses contain some kind of randomness, either in the occupation of the sites or the bonds. We consider a one-dimensional lattice with a classical Hamiltonian

$$E = -\sum_{i<j} J_{ij} s_i s_j. \tag{8}$$

Every site is occupied but the bonds are randomly distributed according to a Gaussian distribution [53]

$$J_{ij} \sim \mathcal{N}\left(0, \frac{C}{r_{ij}^{2\sigma}}\right) \qquad \text{with} \qquad \mathcal{N}(\mu, \sigma^2) \coloneqq \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(x-\mu)^2}{2\sigma^2}\right).$$

The mean μ is zero and the variance σ^2 decays as a power-law of the distance between the spins $r_{ij} = |i - j|$. The parameter σ controls the range of the interactions. Interactions can be ferromagnetic $(J_{ij} > 0)$ or antiferromagnetic $(J_{ij} < 0)$, which causes tensions between alignment and anti-alignment tendencies for the spins. This so-called *frustration* creates disordered metastable states below a critical temperature T_c. This is called the *spin glass phase*: cooperative behaviour between the spins freeze the system at finite temperatures and cause frustration in the ground states [35]. For $T > T_c$, the system exhibits typical magnetic behaviour such as paramagnetism.

The Hamiltonian in (8) has a rich phase diagram, which is determined by the range of the interactions σ. While for $\sigma > 1$ there is no phase transition, for $\frac{1}{2} < \sigma < 1$ the system exhibits a phase transition at $T_c > 0$. More specific, $\frac{1}{2} < \sigma < \frac{2}{3}$ corresponds to mean field theory and $\frac{2}{3} < \sigma < 1$ with an infrared divergence. When $\sigma = 1$, the system has a *Kosterlitz-Thouless* or $T_c = 0$ transition [27].

One-dimensional spin glasses with long-range interactions can be used as a proxy for short-range spin glasses, where the interactions are only between the nearest neighbours. In particular, varying the σ in long-range models corresponds to varying the dimension d in short-range spin glasses [57]. For $\frac{1}{2} < \sigma < \frac{2}{3}$, there is an exact relation between the two [3]

$$d = \frac{2}{2\sigma - 1}. \tag{9}$$

This implies that for $d > d_u = 6$ short-range spin glasses are described by mean-field theory. The dimension d_u is called the *upper critical dimension*. In practice, it is difficult to study systems in high dimensions since the number of spins scales as N^d, with N the linear system size. For example, a range of N-values is necessary to perform *finite-size scaling* around the phase transition to determine the critical temperature T_c. The advantage of the one-dimensional long-range model is that it can be studied for a large range of system sizes and every value of σ [3].

Consider a short-range model in the mean field regime $(d > d_u)$ and spin glass phase $(T < T_c)$. From *Replica Symmetry Breaking (RSB) theory*, the averaged square of the *disconnected spin-spin correlations* $C(i,j) = \langle s_i s_j \rangle$ is found to decay as a power-law with the distance [40]

$$C_4(i,j) \coloneqq \left[\left\langle s_i s_j \right\rangle^2\right] \propto \frac{1}{r_{ij}^{\alpha_s}} \qquad \text{with} \qquad \alpha_s = d-4 \qquad \text{(short-range)}.$$

The average $\langle \ldots \rangle$ is taken over different spin configurations and the average $[\ldots]$ over different bond realisations J_{ij} [53]. If α_s is an exponent for a d-dimensional short-range model, then the corresponding exponent in a long-range model is [3]

$$\alpha_l = \frac{\alpha_s}{d} = 3 - 4\sigma \qquad \text{(long-range)}, \tag{10}$$

where we have used the relation (9). We can use (10) to study the correlations in a long-range one-dimensional spin glass.

1.4 MONTE CARLO

Monte Carlo is quite an overloaded term, since basically every method involving some kind of randomness, is branded as a Monte Carlo method. Here, the most important one is the *Markov Chain Monte Carlo* method, but first the traditional *Monte Carlo integration* is briefly discussed.

Monte Carlo integration is a method to estimate integrals: this works especially well for high-dimensional integrals that are often encountered in statistical physics [37, 47]. The basic idea is to write an integral J as an expectation value over a function

$$J = \int_\Omega f(x)\,dx = \int_\Omega \frac{f(x)}{p(x)} p(x)\,dx \coloneqq \int_\Omega h(x)p(x)\,dx = \langle h(x) \rangle_{x\sim p}\,, \tag{11}$$

where the function p has to represent a probability distribution over the sample space Ω. It follows that the integral or expectation value of the function $h(x)$ can simply be estimated by the sample mean

$$\widehat{J}_K = \frac{1}{K} \sum_{k=1}^{K} h(x_k),$$

where $\{x_k\}_{k\geqslant 1}$ is a sequence of independent random variables distributed according to the distribution p, denoted as $x_k \sim p$. By the strong law of large numbers, the sample mean converges to the integral for $K \to \infty$ with probability one; the estimation is also unbiased: $\left\langle \widehat{J}_K \right\rangle = J$. It is almost always important to have an idea of how good the estimate is. Since the random variables x_k are independent, it is easily shown that the variance of the estimate scales as

$$\operatorname{Var}\left(\widehat{J}_K\right) = \frac{\operatorname{Var}\left(h(x)\right)}{K},$$

which can be estimated by

$$\widehat{\operatorname{Var}}\left(\widehat{J}_K\right) = \frac{1}{K^2}\sum_{k=1}^{K}\left(h(x_k) - \widehat{J}_K\right)^2. \tag{12}$$

For independent random variables, the standard deviation thus scales as $1/\sqrt{K}$. In order to reduce the error, it is possible to choose $p(x)$ so that $\operatorname{Var}(h(x))$ decreases. This variance reducing technique is called *importance sampling*: essentially it comes down to choosing the distribution p as similar as possible to the function f.

Comparing the expression for averages in a canonical ensemble from (3) with the MC average in (11), it is easily seen that it is possible to estimate averages using importance sampling with the already properly normalised Boltzmann distribution from (1). Thus the averages are estimated by sampling K independent configurations $x_k \sim \exp(-\beta E(x_k))$ according to the Boltzmann distribution and calculating

$$\widehat{O}_K = \frac{1}{K}\sum_{k=1}^{K} O(x_k). \tag{13}$$

So in practice, only two things are needed to obtain good estimates of macroscopic quantities in complex physical systems: (1) independent configurations to exploit statistical analysis with independent random variables and (2) a lot of configurations to increase the accuracy of the estimation and to decrease the error.

This is exactly what computational techniques such as Molecular Dynamics and Markov Chain Monte Carlo were developed for: to efficiently sample a lot of configurations distributed according to the Boltzmann distribution. As mentioned before, we only focus on the latter.

1.5 MARKOV CHAIN MONTE CARLO

In the context of statistical physics, x_k represents a state of the system, for example a certain configuration of spins in a magnetic system.

A *Markov chain* is a sequence of dependent random variables $\{x_k\}_{k\geqslant 1}$ that describes a path in a state space where the probability of a state x_k only depends on the previous state x_{k-1}. A Markov chain is completely characterised by the transition probability from one state to another $T(x \to x')$ and the initial probability distribution $\pi(x)$. It can be proven that an *ergodic* Markov chain (*irreducible* and *aperiodic*) converges to a stationary or invariant distribution $p(x)$.

Markov Chain Monte Carlo (MCMC) essentially engineers the transition probability so that it converges to a pre-defined probability distribution $p(x)$ [37, 47]. One possible and popular solution is to choose the transition probability so that it satisfies the *detailed balance equation*

$$T(x \to x')p(x) = T(x' \to x)p(x').$$

Choosing the invariant distribution $p(x)$ as the Boltzmann distribution suffices to sample configurations x from a canonical physical system. This requires an initialisation of the Markov chain and an equilibration or burn-in period until the process has approximately reached its stationary distribution. Only then one can start gathering samples from the Boltzmann distribution.

One of the most widely used practical implementations of the principle of detailed balance is the *Metropolis-Hastings algorithm*. It proposes the following steps to simulate a Markov chain with invariant distribution $p(x)$ [47]:

1. *Initialisation*: choose an initial state x_0.
2. *Trial step*: given a current state x, propose a new state x' with trial step probability $\omega_{xx'}$.
3. *Acceptance step*: compare the two states and accept with probability $A_{xx'} = \min(1, p(x')/p(x))$.
4. *Iteration*: go back to step 2.

After an initial number of steps, the configurations generated in the loop are distributed according to the Boltzmann distribution. These can then be used to estimate macroscopic quantities using (13).

The major advantage of MCMC is that it is well suited to tackle high-dimensional problems as frequently encountered in statistical physics. Its guaranteed convergence to the required quantities makes MCMC a robust algorithm that can be utilised for many different systems [42]. Further, for most systems an MCMC implementation is relatively easy as demonstrated by the Ising model (see Appendix A.1). Third, there is no need to estimate the partition function separately as only ratios of probabilities are considered in the algorithm, which filters out the normalising constant. This saves a lot of compute time which otherwise is necessary to calculate or estimate the partition function. Instead, averages can be estimated directly.

However, MCMC also faces some significant challenges, mostly related to the auto-correlations between subsequent samples of the system, which are introduced when simulating the Markov chain. If a sample is taken at every step, then subsequent samples are highly correlated. This is not necessarily a problem for the estimate (13) itself, but it is for the variance of the estimate. Similarly as before, one

can calculate this variance, but now for correlated random variables as

$$\operatorname{Var}\left(\widehat{O}_K\right) = \frac{\operatorname{Var}(O(x))}{K} + \sum_{k=1}^{K-1} \frac{2(K-k)}{K^2} \operatorname{Cov}(O(x_k), O(x_0)) . \quad (14)$$

When using correlated instead of independent samples, the error of the estimate increases because of the second term. Moreover, if the correlations increase, the sum has larger contributions and the variance increases even more.

Here the correlation time means the number of steps to obtain approximately independent configurations.

Luckily the solution is quite simple: find the correlation time, for example by explicitly calculating it, and only take independent samples to estimate system averages or use another trick such as the *batch means method*. Unfortunately, this means that the simulation has to run for much longer and is computationally much more expensive.

In some cases, the correlation time can even diverge. A well-known example is the divergence of the correlation time near the critical point of the infinite two-dimensional Ising model; this phenomenon is called *critical slowing down*. Around the critical point, the correlation time τ is governed by a dynamical exponent z, $\tau \sim L^z$, with L the size of the system. Even for finite systems, the standard Metropolis algorithm suffers from long correlation times because it has quite a large dynamical exponent. When the system is highly correlated, the updates of the Metropolis algorithm are often rejected. One option is to develop alternative MCMC algorithms to decrease z, such as the cluster-flipping *Wolff algorithm* for the Ising model [54]. In fact, for many spin systems and some off-lattice systems there exist cluster algorithms, which circumvent many of the problems encountered with local updating schemes [18, 31]. Unfortunately, these are not always able to completely remove the critical slowing down and not for all systems a cluster version has been found yet. It is also possible to study the system through *finite-size scaling* by exploiting the scaling of finite-size effects with increasing system size. This general technique finds *critical exponents* that characterise the behaviour of physical quantities, but only works in a narrow region around the critical point [7, 37].

Second, many MCMC algorithms explore the state space by performing local updates which can cause slow convergence [42]. The Markov chain can get stuck in a local minimum of the free energy of the system, preventing it from reaching the true global minimum. For example, this can happen when initiating a two-dimensional Ising system at a temperature smaller than the critical temperature $T < T_c$, using a completely random spin configuration ($T \to \infty$). The lattice can form large regions of up and down spins and it is difficult for one to dominate over the other: this metastable state is an example of a local minimum and is generally something that should be avoided

[37, 47]. This is especially challenging for complex systems with many metastable states such as spin glasses [8]. The problem of slow convergence can, for example, be mitigated by designing algorithms that take into account the global structure of the state space. However, these new techniques introduce new complications [42]. An often-used technique for spin glasses is *parallel tempering*. Multiple copies of the system are run at different temperatures and replica samples are used to update the Markov chain [32]. However, proper equilibration of the system is a challenging issue [23].

Finally, observables that directly depend on the partition function, such as the free energy and entropy from Table 1, are not easily accessible because the MCMC method does not provide the partition function explicitly [38].

2

MACHINE LEARNING

2.1 MACHINE LEARNING

In the real world, many problems and tasks are too difficult to solve by explicitly programming an algorithm. For example, it is very hard to engineer algorithms to recognise faces because a face in itself is very complicated and has a lot of details. Likewise, designing a music recommending system is difficult. Despite knowing the input, namely the music you have listened to in the past, and the output, music that you would like to listen to, it is not immediately clear how to go from the input to the output. These type of problems are present in all aspects and corners of society [2]. In fact, high-dimensional and non-trivial problems are frequently encountered in physics.

In the past decades, *Machine Learning (ML)* has been the most popular method to tackle these problems. Essentially, Machine Learning is learning computers to perform complex tasks or solve complex problems without explicitly programming an algorithm, but instead by learning from data or past experiences [2]. More specific, an ML approach starts by choosing a mathematical model to solve the problem; this model is then optimised by a learning algorithm that minimises a *cost* or *loss function*.

As an example, consider a classification model that should differentiate cat pictures from dog pictures. To train this model, it is necessary to gather pictures of cats and dogs: these form input vectors for the model $\{x_i\}_{i\geqslant 1}$. The corresponding true output labels $\{y_i\}_{i\geqslant 1}$ tell us if each picture x_i is either a cat ($y_i = 0$) or a dog ($y_i = 1$). The mathematical model $g(x;\theta)$ that classifies the pictures usually depends on some parameters θ. The model then guesses the label of a picture x_i: this is the *hypothesis* $\widehat{y}_i \coloneqq g(x_i;\theta)$.

Finding the best model consists of finding the best parameters θ that minimise a well-chosen loss function $L(\theta)$, which often compares the predicted labels $\widehat{y}_i$ with the true labels y_i and is of the form $L(\widehat{y}_i, y_i)$. For a binary classification problem this is typically the *cross-entropy loss*. Usually the loss function is an expectation value over the underlying distribution of the the problem $p(x, y)$:

$$L(\theta) = \langle L(g(x;\theta), y) \rangle_{x\sim p} \,. \tag{15}$$

The main goal of ML models is *generalisation*: the fact that the model generalises to and performs well on unseen data; or in other words, that it can classify pictures of cats and dogs that it has not seen yet. To

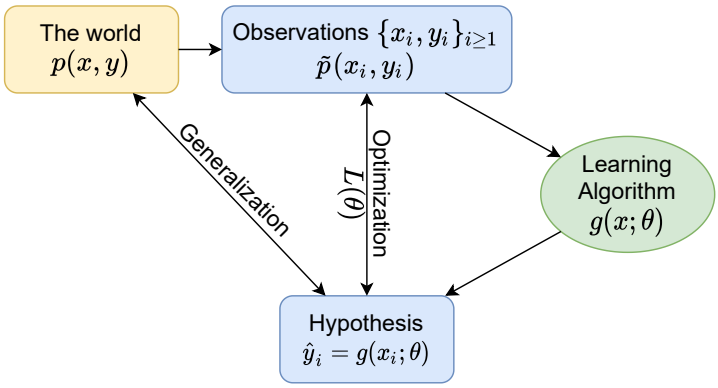

Figure 2: *Empirical Risk Minimization as a Machine Learning framework, based on [19]. Some finite data set* $\{x_i, y_i\}_{i\geqslant 1}$ *is taken from the real world and is used to train a model* $g(x;\theta)$*. This model is then optimised by minimising a well-chosen loss function* $L(\theta)$ *between the hypothesis* $\widehat{y}_i$ *and the truth* y_i*.*

obtain the best generalisation of the model, the loss in (15) should be used: this quantity is called the *risk*. Unfortunately, since the distribution $p(x, y)$ of the problem is generally unknown, the risk cannot be used directly. The best next thing is to use the underlying distribution of the finite training set of samples $\widetilde{p}(x_i, y_i)$:

$$L(\theta) = \langle L\left(g(x_i;\theta), y_i\right)\rangle_{x_i\sim\widetilde{p}}\,. \tag{16}$$

The loss function is then minimised as an expectation value over the distribution of the training set: (16) is the *empirical risk*. The above-described framework is known as *Empirical Risk Minimization (ERM)* and is often used in traditional ML [19]. In Figure 2 the general flow of the framework is shown.

If the expectation across models of the generalisation error is far from the *ground truth*, the model is said to have a high *model bias*. This suggests that you either have bad data, which means that the data does not contain the necessary information to solve the problem, or that the model is not sufficiently complex to describe the information in the data. In the latter case, we say that the model is *underfitting* to the data. If the generalisation error of a model is very sensitive to the variability in the data set, the model is said to have a high *model variance*. This suggests that the model is zooming in on the details unique to that specific data set: the model is *overfitting* to the data. This can be prevented by training the model on more data or performing *regularisation*, which are specialised ML methods designed to reduce overfitting. In ML it is typically difficult to train a model which performs well on unseen data (low bias) and is not sensitive to the training data (low variance). The best model is often a trade-off between a low bias–high variance or high bias–low variance model: this is called the *bias-variance trade-off*. In practice, it is very common

to start from a complex model with many parameters and properly regularise it to reduce overfitting and improve generalisation [19].

Machine Learning can be used to sample configurations from the Boltzmann distribution of physical systems. This can be done by learning the Boltzmann distribution itself. If the total probability distribution $p(x)$ is known, it is straightforward to sample configurations x from this distribution by sequentially generating samples x with probability $p(x)$. The ML problem then becomes a *density estimation* problem: given a data set of configurations of the system x, estimate the density of the samples $p(x)$. If the data set is a good representation of all possible configurations of the system, the learnt estimated distribution is a good approximation of the distribution of the system. Note that it is already hinted that ML is heavily data-driven: that means that the quality of the data set is very important to obtain good final results.

There are many different types of ML models, but here only the relevant ones are discussed:

- *Supervised model*: the data is observed and labelled by corresponding targets. In this case, it is possible to model a direct relationship between input x and output y. For example, an image classification problem where both the images and the class – cat or dog – are given, is supervised.

- *Unsupervised model*: the data is observed but unlabelled, so it is only possible to model the underlying structure of the data.

The definition of unsupervised ML leans very closely to density estimation: when there is no desirable output available, the only thing left is to learn something about the underlying structure or density of the data. For density estimation, the exact labels $p(x)$ are not known, since this is exactly what you are trying to learn. Learning the Boltzmann distribution is thus generally unsupervised and requires techniques developed in unsupervised ML.

As an example, consider a data set $\{x_i\}_{i\geqslant 1}$ of spin configurations belonging to a one-dimensional Ising system with N spins, where each data point $x_i = (s_1^i, \ldots, s_N^i)$ represents a spin configuration. The underlying distribution of the data set must be the Boltzmann distribution: configurations with high Boltzmann probability should appear more often than configurations with low probability. In fact, each sample x should occur with a probability very close to the real Boltzmann probability $p(x)$. The idea is to train a model $g(x_i;\theta)$ by minimising a loss function, so that the output of the model is a good estimate for the Boltzmann probability: $g(x_i;\theta) \approx p(x_i)$.

Not only the distinction between supervised and unsupervised is important here, but also between *discriminative* and *generative* models:

- *Discriminative model*: it is possible to learn the posterior distribution $p(y \mid x)$ directly, either implicitly or explicitly. For example, a discriminative classification model directly models the probability of the class (cat or dog) given an input image, e.g. $p(\text{dog} \mid \text{image of a cat})$.

- *Generative model*: model the joint probability of the input and labels $p(x, y)$ and use probability theory to obtain the probability of labels given the data. For example, a generative classification model models the joint probability of the class (cat or dog) and the input image, e.g. $p(\text{dog}, \text{image of a dog})$. One can then use *Bayes theorem* to get the posterior probability as in the discriminative case:

$$p(y \mid x) = \frac{p(x, y)}{p(x)}.$$

  The most important advantage of generative models is that they allow us to generate new data (x, y); in this case, it is possible to create new images of cats and dogs.

Since the goal is to generate new samples from the Boltzmann distribution, generative models are the better choice since they allow us to effectively generate new samples x with probability $p(x)$.

In summary, we use unsupervised generative modelling to estimate the density of the Boltzmann distribution using ML techniques. The learnt distribution can then be used to sample configurations from the physical system, as an alternative to previously mentioned methods such as Markov Chain Monte Carlo. The next step is to choose an appropriate ML model.

2.2 GENERATIVE MODELS

There are many options in generative modelling to learn the distribution of physical systems $p(\mathbf{x})$, each with its own strengths and weaknesses. Examples are *Boltzmann Machines* [10, 13, 34, 48], *Normalising Flows* [1] and *Generative Adversarial Networks* [30]; [51] gives a brief overview. A summary of commonly used generative models can be found in Table 2. Here we focus on *Autoregressive Models* [21, 55] because of their simplicity and the ability to provide a tractable likelihood $q(\mathbf{x}; \theta)$, which allows direct sampling of new configurations. Autoregressive models also work for both discrete [45] and continuous [52] sample spaces.

Consider a system which consists of N variables x_i so that a single configuration of the system can be expressed as $\mathbf{x} = (x_1, \ldots, x_N)$. Us-

GENERATIVE MODEL	TRACTABLE LIKELIHOOD	DISCRETE SAMPLE SPACE
Boltzmann Machines	✗	✓
Autoregressive Models	✓	✓
Normalising Flow	✓	✗
Variational Auto-encoders	✗	✓
Tensor Networks	✓	✓
Generative Adversarial Networks	✗	✓

Table 2: *A summary of some frequently used probabilistic generative models in physics, based on [19, 51]. The table also shows if the model has explicit access to a tractable likelihood of the generated samples, and if the model can be used for discrete sample spaces such as the Ising model.*

ing the definition of conditional probabilities, the probability of a configuration of the system can be written as

$$p(\mathbf{x}) = \prod_i p(x_i \mid \mathbf{x}_{<i}) = \prod_i p(x_i \mid x_{i-1}, \ldots, x_1). \tag{17}$$

This is called the *autoregressive property*. Instead of one model learning the entire joint distribution $p(\mathbf{x})$, each conditional $p(x_i \mid \mathbf{x}_{<i})$ is separately parameterised by an ML model. For physical systems in general, such a representation grows exponentially with system size. For example, if each variable x_i is binary and can take on two values, a full description of the joint distribution typically requires around 2^N parameters [19]. However, systems in the real world often display some kind of structure on the variables x_i, which makes it possible to parameterise the joint distribution with less than an exponentially growing number of parameters [21].

In combination with ML principles such as sharing and reusing the same parameters across the model, it is possible to learn each of the conditionals without requiring an exponentially growing number of parameters. Although the number of parameters is limited, a description of each conditional separately is still able to capture high-order dependencies between the random variables x_i. Once each conditional probability is learnt, they can be chained together to generate new samples of the system $\mathbf{x}$ [49].

2.3 DEEP LEARNING

Traditional Machine Learning is focused on *feature engineering*: using algorithmic approaches to cleverly extract the information from the

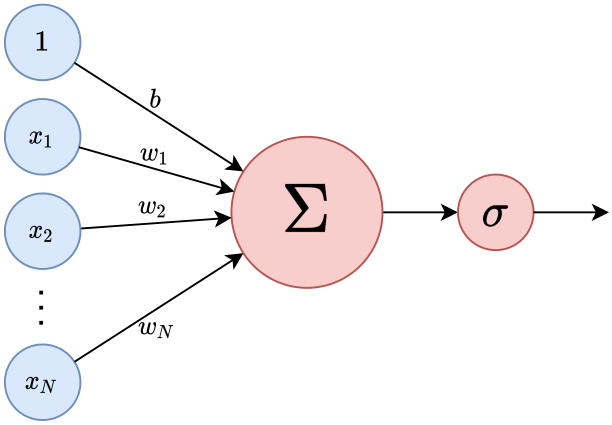

Figure 3: *A graphical representation of a perceptron: a weighted sum Σ of an input vector $\mathbf{x}$ is passed through a non-linear activation function σ.*

data which is needed to solve the problem. While this approach has been adequate to solve some difficult tasks, such as image recognition, at some point the human cleverness reaches its limit: there are many tasks in which the raw data is not easily transformed into useful information which a model can use to solve the problem. These type of problems is exactly where the field of *Deep Learning (DL)* performs well.

Deep Learning is based on a specific type of ML model that recently has been very successful in solving highly complex problems, namely a *Neural Network (NN)*. Instead of putting the complexity of the problem in the data and features, it moves the complexity to the model. That means that the model itself learns to extract the relevant information from the data and no feature engineering is required. This behaviour is called *feature learning* since NNs can learn to extract the necessary features from the raw data [26].

The term perceptron is quite overloaded; we consider the more general definition where the activation function can be pretty much any non-linear function.

The building blocks of NNs are *perceptrons*: a weighted sum that is sent through a non-linearity. To clarify, consider an input vector $\mathbf{x} = (x_1, \ldots, x_N)$; a perceptron model calculates a weighted sum Σ of the input vectors

$$\Sigma(\mathbf{x}) = \sum_{i=1}^{N} w_i x_i + b,$$

with the weights w_i and bias b as parameters of the model. The output of this weighted sum is sent through a non-linear *activation function* σ, which is often a hyperbolic tangent or sigmoid. The total output of the perceptron is

$$g(\mathbf{x}; \mathbf{w}, b) = \sigma\left(\Sigma(\mathbf{x})\right) = \sigma\left(\sum_{i=1}^{N} w_i x_i + b\right). \tag{18}$$

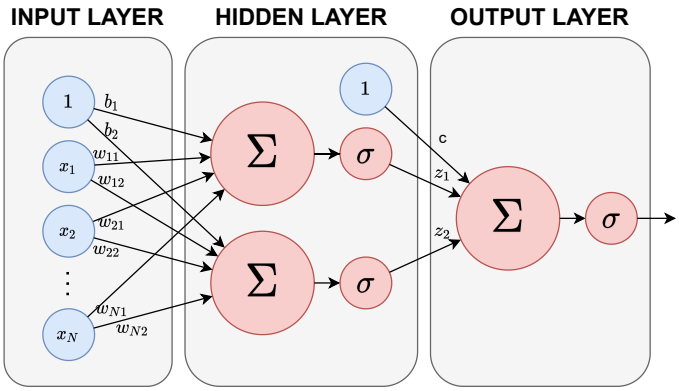

Figure 4: *A graphical representation of a Feedforward Neural Network with an input layer, one hidden layer containing two hidden nodes and an output layer. The parameters of the network are the weight matrix* W_{ij}*, weight vector* $\mathbf{z}$ *and corresponding biases* $\mathbf{b}$ *and* c*.*

Figure 3 depicts a graphical representation of a perceptron. An NN is built by combining multiple perceptrons. For example, if you take multiple weighted sums of the input vector $\mathbf{x}$ and pass each one through an activation function, then the outputs can be used as the input of a third perceptron. This construction is an NN with one hidden layer containing three hidden nodes and can be seen in Figure 4.

If one keeps adding nodes to the hidden layer, it is possible to approximate any function or boundary: this is known as the *universal approximation*. However, the computational expense for training builds up quite rapidly with only one hidden layer containing many nodes. In practice, it is possible to get very good results by introducing more hidden layers, each with a limited number of hidden nodes, while still being computationally feasible. By introducing more layers, the network can reach higher complexity with fewer nodes in total [19].

The previous examples of Neural Networks do not contain any loops: the information flows in only one direction, starting from the input through the hidden layers to the output, without cycling back to previously visited nodes. This network type is called a *Feedforward Neural Network (FFNN)*. There also exist network architectures that do allow cycles and loops in the information flow; these are introduced further down. An example of an FFNN with two hidden layers, containing two and three hidden nodes respectively, can be seen in Figure 5.

The activation function must be non-linear, otherwise, the model is just chaining multiple linear operations together, which is also linear. The power of NNs lies in the high complexity of the models that can only be introduced by non-linear functions.

Neural Networks as a composition of weighted sums and non-linearities may seem arbitrary at first, but apparently, it introduces

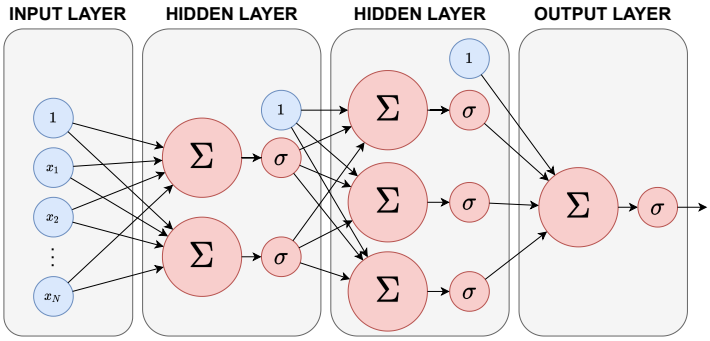

Figure 5: *A graphical representation of a Feedforward Neural Network with an input layer, two hidden layers containing two and three hidden nodes respectively, and an output layer.*

the right kind of complexity needed to solve really difficult problems. This is one of the major advantages of DL [26].

Another advantage of DL is the general mechanism that can be used to optimise most NNs, namely *gradient descent*. Remember that ML solves a problem by finding the model parameters θ that minimise a loss. The best model is the one with the smallest loss, which ideally corresponds to a global minimum of the loss function $L(\theta)$. For a convex loss function, any local minimum corresponds to a global minimum, and the optimisation is relatively easy. A typical example of a convex loss is the *Mean Squared Error (MSE)*

$$\mathrm{MSE}(\theta) = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)\,,$$

which is a sum over the squared differences between the true labels y_i and the predicted labels $\hat{y}_i$ of the data points x_i. Unfortunately, some loss functions can be non-convex: the loss landscape can have multiple local minima, saddle points or flat regions. In this case, it is much more difficult to find a global minimum of the loss and in practice, one often settles for a decent local minimum. In particular, loss functions for deep NNs are often non-convex and thus harder to optimise [19].

Gradient descent is a local minimisation procedure, where the parameters are optimised by taking small steps in the direction of a smaller loss value. This is done by updating the parameters in the negative direction of the gradient of the loss. To clarify, consider the weights W_{ij} as parameters of an NN. In many cases the error E is a function of the output of the network $g(x; W_{ij})$, which in turn is a function of the weights of the model: $E(g(x; W_{ij}))$. Since we know that the output of the network is just a composition of weighted sums and non-linear activation functions, the derivatives of the error to the parameters are differentiable:

For example, think back to the output of a perceptron (18).

$$\frac{\partial E}{\partial W_{ij}} = \frac{\partial E}{\partial g}\frac{\partial g}{\partial W_{ij}},$$

where the second derivative can be calculated using the chain rule on weighted sums and activations. Of course, this only works if the activation functions are differentiable. The first derivative often contains a sum over all data points: this is the traditional gradient descent. Recently, it has been shown that it is often better to introduce stochasticity into the update rule by only calculating the gradient using a random fraction of the total data set: this is called *mini-batch stochastic gradient descent*, or often just *Stochastic Gradient Descent (SGD)*. The stochasticity helps the algorithm to avoid getting stuck in local minima [26].

Once the gradient $\nabla_W E$ is calculated, the weights are updated using a simple rule

$$W_{ij} \to W_{ij} - \eta \frac{\partial E}{\partial W_{ij}}, \tag{19}$$

where the *learning rate* η decides how large the steps are. Not only is gradient descent a general mechanism that can be applied to many different loss functions, but there also exists a very efficient method to calculate the gradients: *back-propagation*. Back-propagation libraries can be combined with ever-increasing compute power (GPUs) to efficiently train deep NNs with many layers containing many nodes, which was before impossible due to the sheer scope of the calculations [19, 26].

2.4 STRUCTURED DATA

For an FFNN, where all nodes between two layers are connected, the number of weights – and thus parameters of the model – quickly increases. For example, for a network with only one hidden layer containing h nodes, input vector $\mathbf{x} = (x_1, \ldots, x_N)$ and output vector $\mathbf{y} = (y_1, \ldots, y_M)$, the number of parameters is

$$P_\theta = ((N \times h) + h) + ((h \times M) + M).$$

Optimising many parameters is often difficult because the model tends to overfit to the data and the training can become computationally expensive. However, in the real world data tends to contain some kind of structure and this structure can be exploited to limit the number of parameters and reduce the chances of overfitting. For example, images are structured two-dimensional arrays of pixels where neighbouring pixels are highly correlated. Similar, a signal or a time

sequence is a one-dimensional array of inputs where inputs close to each other are highly correlated [26].

Deep Learning models limit the number of parameters by sharing or applying the same parameters over different parts of the network. This is actually an important concept in DL which makes training networks for complex problems feasible. There are two major NN types that use this principle:

- *Convolutional Neural Network (CNN)*: real-world data is often composed of a hierarchy; multiple lower level features form higher-level features. For example in images, edges and shapes combine to form a finger, multiple fingers form a hand and a hand is a part of the body. Convolutional Neural Networks exploit this grid-based structure by combining *convolutional layers* and *pooling layers*. Convolutional filters can detect local motifs in the image. Since these motifs can typically appear anywhere in the image, it is possible to use a convolutional filter with the same weights applied to different parts of the image. This weight sharing drastically reduces the number of parameters of a network. Pooling layers coarse-grain the information of motifs into a more dense representation, which is independent of small shifts. For example, two hands can differ slightly due to different rotations or finger placement. The pooling layer tries to transform all hands into the same compact representation. The convolutional and pooling layers are stacked and typically followed by fully connected layers. Training a CNN can be done through the same back-propagation technique [26].

- *Recurrent Neural Network (RNN)*: this is a network that is based on FFNNs but includes weight sharing and feedback loops. Recurrent Neural Networks are designed to handle sequential data, such as time sequences. The network has a memory unit bookkeeping important information and uses it as input for the next layers. They can model dynamic behaviour in the data and make predictions on what might happen next. An example of an architecture can be seen in Figure 6. Recurrent Neural Networks are trained using *back-propagation through time*, a similar algorithm as the standard back-propagation. Table 3 gives an overview of some important advantages and disadvantages of RNNs as DL models [19].

2.5 MACHINE LEARNING IN PHYSICS

Machine Learning and statistical physics are both trying to solve complex problems. This often concerns high-dimensional data, for which the sheer dimensionality and size make the problem very challen-

ADVANTAGES	DISADVANTAGES
Possible to process any input length	Training is sequential and can be slow
Parameters can be shared over the input	Difficult to remember long-term dependencies
Able to take into account past information (memory)	Not able to take into account future information

Table 3: *An overview of some important advantages and disadvantages of RNNs as DL models, based on [14, 19]*

ging [9, 11]. This is known as *the curse of dimensionality* [19]. In particular, the state space of physical systems often scales exponentially with system size [11]. Machine Learning has developed techniques to characterise and handle high-dimensional data but is typically more focused on performance rather than interpretation. In physics, they are generally more interested in interpreting the underlying mechanisms that drive processes. In this way, the ML and physics community might help each other which could result in improvements for both fields [9].

For example, CNNs are well suited to extract high-level features from low-level data. This is very similar to detecting and extracting topological phases and phase transitions from configurations of physical systems [5, 11, 52]. Second, ML models have been used for building representations of challenging quantum many-body ground states, both using CNNs [28] and RNNs [13, 21, 28]. Third, similar to standard MC simulations, NNs can be used to simulate configurations of physical systems. Both CNNs [30, 45, 55, 56] and RNNs [21, 43, 46] have been successfully utilised as architectures to sample configurations and estimate physical quantities. In this study, autoregressive models are combined with RNNs as generative models to learn the Boltzmann distribution of physical systems and generate new configurations. These configurations can then be used to reliably estimate physical quantities.

Part II

RESEARCH

3 LEARNING WITH DATA

3.1 ONE-DIMENSIONAL ISING

The goal is to learn the Boltzmann distribution from (1) using generative ML models to efficiently sample new configurations and study the behaviour and emergent features of physical systems.

Let us start by choosing a relatively simple physical system, namely the one-dimensional Ising system described in chapter 1. Consider a chain of N spins with open boundary conditions. As mentioned before, ML is primarily data-driven. To learn the Boltzmann distribution, it is necessary to start from existing configurations of the system. The most straightforward method is to generate these configurations using the MCMC technique, discussed in section 1.5. The idea is to generate a data set consisting of n samples, each a spin configuration $\mathbf{s} = (s_1, \ldots, s_N)$, and to estimate the underlying density – which is the Boltzmann distribution – of the data set.

For the one-dimensional Ising model, we consider the physical properties from Table 1. Since MCMC does not provide the partition function explicitly, the free energy and the entropy can not be estimated directly. The other quantities can simply be estimated by ensemble averages as in (3); for these, you do not need the partition function. The details about the MCMC algorithm and data generation, in general, can be found in Appendix A.1.

3.2 RECURRENT NEURAL NETWORK

Instead of learning the entire Boltzmann distribution at once, the autoregressive property (17) is used to learn each conditional probability separately:

$$p(\mathbf{s}) = \prod_{i=1}^{N} p(s_i \mid \mathbf{s}_{<i}) = \prod_{i=1}^{N} p(s_i \mid s_{i-1}, \ldots, s_1).$$

Since the one-dimensional Ising system is structured as a chain, the configurations can be interpreted as a sequence input for an RNN. The outputs of the RNN are the predictions for the conditional probabilities: $q_\theta(s_i \mid \mathbf{s}_{<i}) \coloneqq \widehat{p}(s_i \mid \mathbf{s}_{<i})$, with θ the parameters of the model.

An RNN can learn the conditionals $p(s_i \mid \mathbf{s}_{<i})$ sequentially using the spins s_i as input and keeping track of the history through a *hidden vector* $\mathbf{h}$. In particular, the network can be designed to predict the next spin s_i, based on the information of all previous spins in the

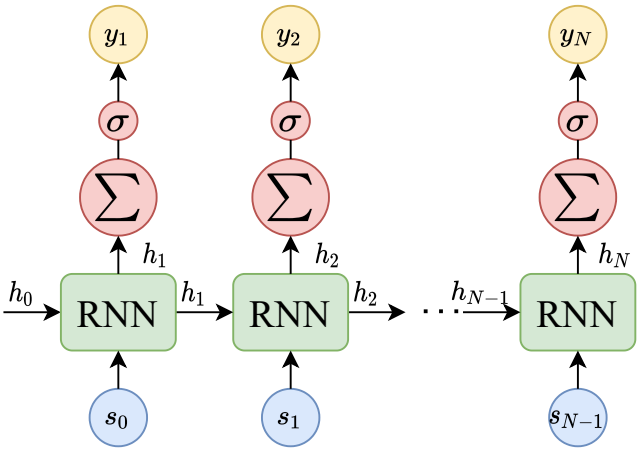

Figure 6: *A graphical representation of the RNN architecture, based on [21]. The network is initialised by the artificial spin s_0 and hidden vector $\mathbf{h}_0$. They are sent through an RNN cell (green box), fully connected layer Σ and softmax activation σ to provide the output probabilities y_i.*

chain $\mathbf{h}_i \coloneqq \mathbf{h}\left(\{s_j\}_{j<i}\right)$. This prediction can easily be encoded into a conditional probability distribution $y_i \coloneqq q_\theta(s_i \mid \mathbf{s}_{<i})$. Once the spins $s_i = -1$ are one-hot encoded into $\mathbf{s}_i = (1,0)$ and $s = +1$ into $\mathbf{s}_i = (0,1)$, each conditional can be calculated as a dot-product between the inputs and the outputs of the network

$$q_\theta(s_i \mid s_{i-1}, \ldots, s_i) = \mathbf{s}_i \cdot \mathbf{y}_i = \mathbf{s}_i \cdot \begin{pmatrix} q_\theta\left(\mathbf{s}_i = (1,0) \mid \mathbf{s}_{<i}\right)) \\ q_\theta\left(\mathbf{s}_i = (0,1) \mid \mathbf{s}_{<i}\right)) \end{pmatrix}.$$

The conditionals are chained together using the autoregressive property to obtain the probability of the full configuration

$$q(\mathbf{s};\theta) = \prod_{i=1}^{N} q_\theta(s_i \mid s_{i-1}, \ldots, s_1) = \prod_{i=1}^{N} \mathbf{s}_i \cdot \mathbf{y}_i. \tag{20}$$

Figure 6 shows a graphical representation of the network architecture; further details about the network can be found in Appendix B.1.

Ultimately, the goal is to use the trained network as a generative model to sample new configurations from the Boltzmann distribution. Sampling is done in a similar sequential fashion as training: once the network is initialised by $\mathbf{s}_0$ and $\mathbf{h}_0$, the conditional probability distribution $\mathbf{y}_1 = q_\theta(s_1)$ is used to generate a spin s_1. This spin is then used as input for the next RNN cell, which outputs $\mathbf{y}_2 = q_\theta(s_2 \mid s_1)$, and is used to sample the second spin in the chain s_2. This procedure is repeated until all spins are sampled and are combined to form one generated configuration $\mathbf{s} = (s_1, \ldots, s_N)$. The sampling procedure is visualised in Figure 7.

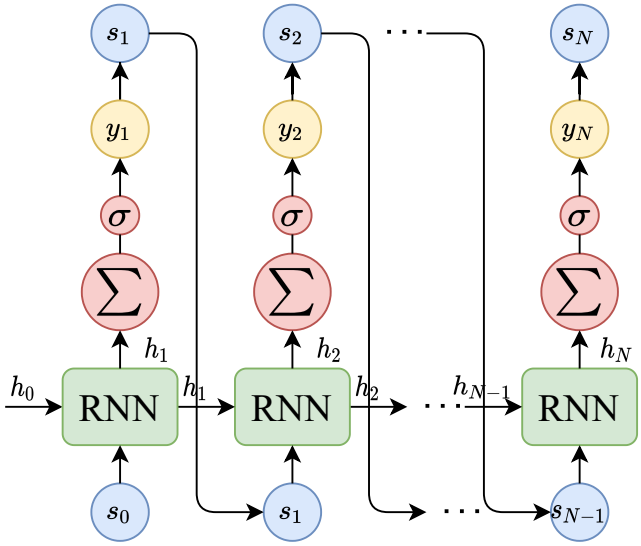

Figure 7: *A graphical representation of the RNN sampling procedure, based on [21]. The output probabilities are used to sequentially generate samples s_i and as input for the next RNN cells.*

For large networks, the chain rule sums multiplications containing many factors, which can be smaller or larger than one. This can lead to instabilities during the training process, known as *vanishing* or *exploding gradients*, which are prevalent in RNNs because the cell operation is repeated multiple times [19]. For physical systems, long-range correlations between the spins can also introduce vanishing or out-of-control gradients. Therefore, the simple RNN cell is replaced by a more complex version, which tries to avoid these complications: a *Gated Recurrent Unit (GRU)* cell [15, 21]. A simple but effective method to avoid exploding gradients is *gradient clipping*: if the gradients of the parameters during training are too large, they are cut at a clipping value. This prevents the network from making very large update steps, which results in unstable learning [19].

The complexity of the network can be easily increased by stacking multiple RNN layers on top of each other. The output of the first RNN cell is then used as input for the next layer. Increasing the hidden dimension – this is the size of the hidden vector – also increases the expressiveness of the model.

A major advantage of training this autoregressive RNN is that the network is not only able to generate new configurations, it also gives the corresponding normalised probability of those configurations. This follows from the fact that the output probabilities $\mathbf{y}_i$ are normalised by the softmax function. As a product of normalised conditional probabilities, the total probability from (20) is also normalised. This is definitely not the case for all generative models: very often it is only possible to generate samples without corresponding probabilities and most often the probabilities are not properly normalised (see Table 2)

[51]. The fact that the samples are generated with normalised probabilities, is beneficial from a physics perspective. More specific, the probabilities can be used to compute the relevant physical quantities [38].

Note that the input sequence of the network is initialised by an artificial spin s_0 and has a final input s_{N-1}. The input chain is open on both sides and resembles an Ising chain with open boundary conditions. Since there is no straightforward method to close the input sequence of an RNN, it is non-trivial to incorporate periodic boundary conditions for the system. Therefore the system under consideration is an Ising chain with open boundaries.

Finally, it is important to understand on a theoretical basis why this architecture might work well for the one-dimensional Ising. The physical properties of the system are entirely based on the classical Hamiltonian (6), where the coupling J is constant throughout the system. Therefore, the physics of the one-dimensional Ising are local. For example, there is no real difference between the $-Js_is_{i+1}$ and $-Js_{i+1}s_{i+2}$ term and ultimately, each term in the Hamiltonian produces the same physical behaviour (not including boundary effects). Because of the local physics, the network has to learn the same information on each lattice site.

Compare this with the chosen architecture of the network: there is only a single RNN cell and fully connected layer (see Figure 6), which are repeatedly applied to each lattice site. While this gives a strong regularising effect by sharing parameters, generally there is no reason to not use a different RNN cell for each lattice site. However, since each RNN cell has to learn the same local physics, we have chosen the cells to be identical and share the same parameters. In terms of the transfer matrix method, there is only one transfer matrix that propagates through the system which the network needs to learn.

On top of that, RNN architectures are well suited to study one-dimensional lattice systems with open boundary conditions. While each lattice site is described by a single RNN cell and fully connected layer, the interactions between sites can be associated with the passing of a hidden vector. For higher-dimensional lattice systems, the RNN can still be used but the transformation of the lattice sites into a one-dimensional sequence is not trivial. For example, a two-dimensional Ising system can be snaked into an input sequence for an RNN [21].In that case, the interactions between sites are not always associated with the direct passing of a hidden vector, which causes local correlations in the two-dimensional lattice to be mapped onto non-local correlations in the one-dimensional input sequence [21].

In summary, while this particular network architecture might work well for the one-dimensional Ising, it is probably not suited to study most other physical systems. The performance of the network can be

increased by implementing the physics of the system into the architecture [22]. We come back to this in chapter 5.

3.3 TRAINING THE NETWORK

Consider a data set of spin configurations $\mathcal{D} = \{\mathbf{s}_i\}_{i \geqslant 1}$, generated by the MCMC algorithm. Once the RNN architecture is built, the model can be optimised by choosing a loss function. A popular metric for distribution estimation is the *Kullback-Leibler (KL) divergence* D_{KL}, which measures the similarity between two distributions [51, 55]. Here, it represents the similarity between the Boltzmann distribution of the system $p(\mathbf{s})$ and the distribution of the network $q(\mathbf{s}; \theta)$ from (20):

$$D_{KL}(p \parallel q) = \sum_{\mathbf{s}} p(\mathbf{s}) \ln\left(\frac{p(\mathbf{s})}{q(\mathbf{s};\theta)}\right) = \left\langle \ln\left(\frac{p(\mathbf{s})}{q(\mathbf{s};\theta)}\right)\right\rangle_{\mathbf{s}\sim p}. \quad (21)$$

It holds that $D_{KL} \geqslant 0$ due to the *Jensen inequality*, where the equality only holds if the two distributions are the same. To optimise the ML model, we want to find the best parameters θ that minimise the KL-divergence. It can be shown that this is equivalent to maximising the likelihood $\ell(\theta)$ of the data [51]. This optimisation technique is called *Maximum Likelihood Estimation (MLE)*, since it maximises the likelihood of occurrence of the samples in the data set:

$$\ell(\theta) = p(\mathcal{D}; \theta) = \prod_{\mathbf{s}\in\mathcal{D}} p(\mathbf{s}; \theta).$$

Note that the factorisation is only possible when the data samples are independent, which means that MLE, in this expression, uses the *i.i.d. assumption*: the configurations are independently and identically distributed from the Boltzmann distribution. In practice, it is more common to minimise the *Negative Log-Likelihood (NLL)*

$$L(\theta) = -\ln(\ell(\theta)) = -\sum_{\mathbf{s}\in\mathcal{D}} \ln(p(\mathbf{s};\theta)). \quad (22)$$

Since the RNN model immediately outputs the probabilities of the samples via (20), the loss function is just (22) but using the outputs of the network $q(\mathbf{s}; \theta)$. For Stochastic Gradient Descent, the loss is typically not calculated for the whole data set, but only for a mini-batch of samples. The training loss from (22) becomes

$$L(\theta) = -\frac{1}{n_b}\sum_{i=1}^{n_b} \ln(q(\mathbf{s}_i;\theta)), \quad (23)$$

where the NLL is divided by the batch size n_b. Since the parameters of the network are updated using n_b training examples, the batch size

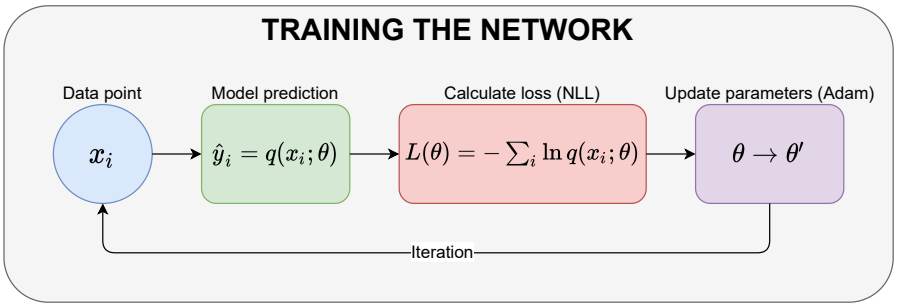

Figure 8: *A flowchart of the training process: a batch of spin configurations x_i is forwarded through the network, which provides the predicted probabilities $\hat{y}_i$ and are used to calculate the NLL loss $L(\theta)$. Finally, a back-propagation through the network provides the gradients of the loss and are used by the Adam optimiser to update the parameters $\theta \to \theta'$. This procedure is repeated until the training has iterated a fixed number of times through the data. We choose the best model based on the parameters that lead to the minimal validation loss.*

impacts the memory requirements during training. Although SGD is a powerful technique, recently it is often replaced by more sophisticated alternatives. We use the *Adam* optimiser, which combines an adaptive learning rate with momentum techniques. It is a more complex version of the normal SGD optimisation, designed to improve efficiency and convergence of the training process [25].

The network can be trained by minimising the NLL loss function (22) and using the Adam optimiser for updating the parameters. Iterating multiple times over the data – one training iteration over the full data set is called an *epoch* – the network tries to find an optimal set of parameters θ so that the output $q(\mathbf{s};\theta)$ approximates the Boltzmann distribution $p(\mathbf{s})$. The training flow is visualised in Figure 8 and the hyperparameters of the network can be found in Table 4. Note that not all parameters are optimised, but some are fixed, based on previous research [21, 55].

Let us consider the effect of the different parameters on the training of the network. First, networks with different model complexity are trained, which is governed by the number of stacked RNN layers and the hidden dimension. A model with a larger number of parameters can typically describe more complex data but is also more sensitive to overfitting. Figure 9 illustrates the impact of the number of parameters on the training of the network for two different system sizes.

For a small system size ($N = 10$), the network starts to overfit to the data, decreasing the training loss and increasing the validation loss, past a certain number of epochs. For a larger system size ($N = 50$), the network displays much less overfitting; only for a very powerful network with many parameters overfitting occurs. The model is regularised because the parameters of the RNN cell and the fully connected

HYPERPARAMETERS	RANGES
Loss function	NLL
Optimiser	Adam
Number of epochs	100
Batch size n_b	$10 - 1000$
Learning rate η	$0.001 - 0.1$
Hidden dimension h_d	$1 - 75$
Number of layers n_l	$1 - 3$
Gradient clipping value c_g	2

Table 4: *The hyperparameters θ of the network that are optimised to find the best solution $q(\mathbf{s};\theta)$. The fixed choices and typical tuning range for the hyperparameters during training are also shown.*

layer are shared over the inputs. This reduces overfitting to the data. However, longer spin chains are less affected by overfitting which means that increasing the system size must introduce some other regularisation effect. This is probably caused by the locality of the one-dimensional Ising system. As mentioned before, the network has to learn the same behaviour at each lattice site. Because longer spin chains have more examples of two neighbouring spins $-Js_is_{i+1}$, the model is less likely to overfit. In this interpretation, more examples of local physics can be seen as having more data available. If an ML model is trained on more data, it is less likely to overfit [19]. Because we consider typical system sizes of $N \gtrsim 100$ and networks with a limited number of parameters ($P \lesssim 1000$), overfitting is not a main concern. Therefore, we do not apply regularisation methods such as *dropout* [19].

For the same data sets, networks with varying batch sizes are trained. The result of the training loss (23) can be found in Figure 10. A small batch size requires less memory and introduces more stochasticity in the loss function while also performing many update steps per epoch. A batch size that is too small often results in poor estimates of the gradients of the loss, which can lead to unstable learning and requires a small learning rate. Training with a very large batch size also seems unstable in comparison to smaller batch sizes. This is caused by the averaging of the loss over one epoch. For $n_b = 1000$, there are only two batches in one epoch while for $n_b = 500$, there are already five batches in one epoch. For smaller systems, $N = 10$, the network can overfit depending on the batch size. For a larger system, $N = 50$, overfitting is much less present and all training curves coincide.

Figure 11 displays the training loss for varying learning rate η. There are no noticeable differences between the training curves for different system sizes. For many DL problems, the learning rate is an

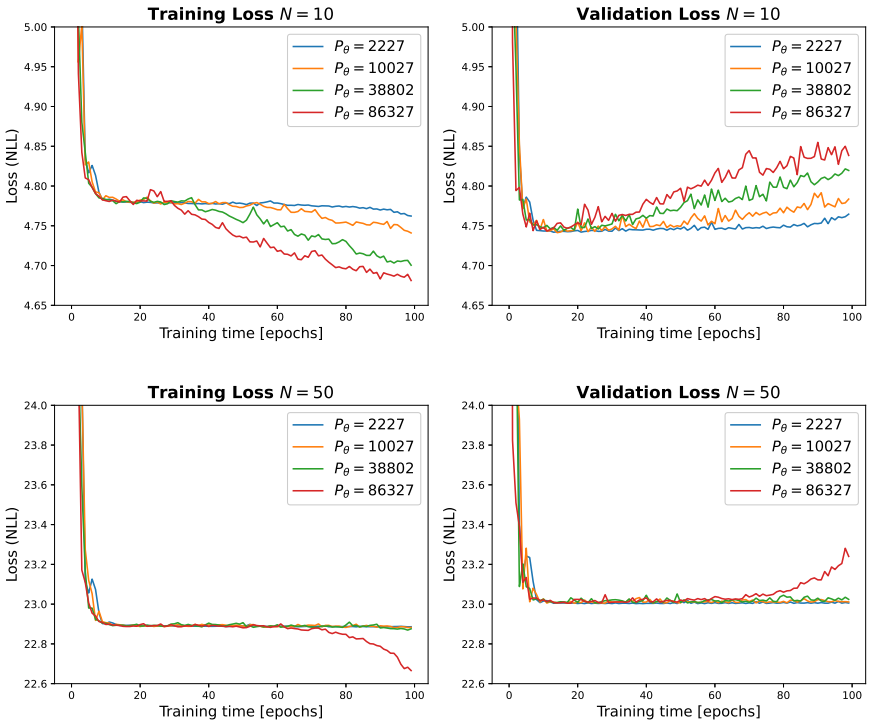

Figure 9: *The training and validation loss curves during training of networks with an increasing number of parameters* P_θ. *The networks are trained on a data set with* $n = 5000$ *samples:* 2500 *train,* 1500 *validate and* 1000 *test samples. The data sets consist of Boltzmann configurations of the one-dimensional Ising with* $N = 10$ *(top) or* $N = 50$ *(bottom) spins.*

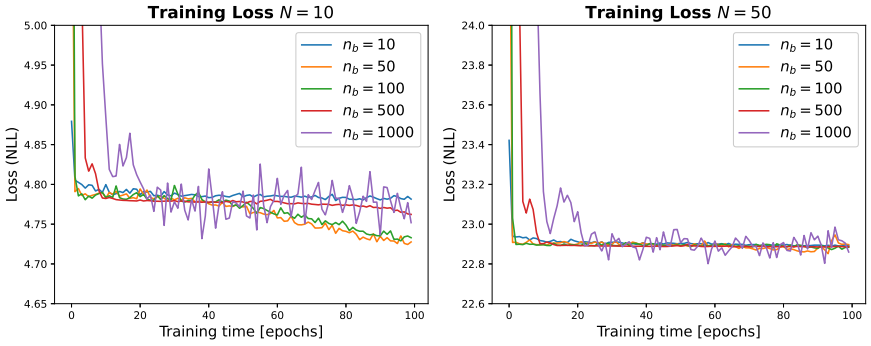

Figure 10: *The training loss curves during training of networks with varying batch sizes* n_b. *The networks are trained on a data set with* $n = 5000$ *samples:* 2500 *train,* 1500 *validate and* 1000 *test samples. The data sets consist of Boltzmann configurations of the one-dimensional Ising with* $N = 10$ *(left) or* $N = 50$ *(right) spins.*

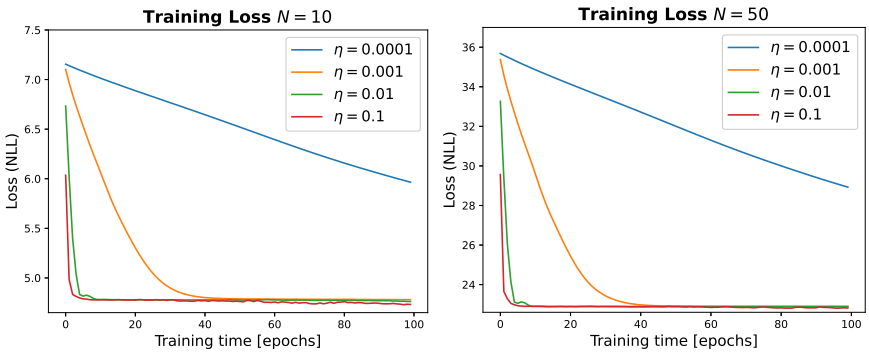

Figure 11: *The training loss curves during training of networks with varying learning rates* η. *The networks are trained on a data set with* $n = 5000$ *samples:* 2500 *train,* 1500 *validate and* 1000 *test samples. The data sets consist of Boltzmann configurations of the one-dimensional Ising with* $N = 10$ *(left) or* $N = 50$ *(right) spins.*

important hyperparameter because of its intricate relationship with the other hyperparameters [26]. However, in this case all networks (except for $\eta = 10^{-4}$) reach approximately the same minimum loss: $4.73 - 4.79$ for $N = 10$ and $22.8 - 22.9$ for $N = 50$. More training does not reduce the training loss substantially. The only major difference is the speed of reaching this minimum value: training with a smaller learning rate is slower because the update steps are smaller. For $\eta = 10^{-4}$ training is very slow and the network has to be trained for much longer to reach the same minimum. This suggests that gradient descent finds approximately the same solution for all values of the learning rate in which case the learning rate only determines the rate of reaching the minimum. Accordingly, the learning rate is not a critical hyperparameter upon training the one-dimensional Ising.

In practice, there are many optimisation methods to find the hyperparameters that lead to a minimal loss. We perform a brute force *grid search* because the number of trainable hyperparameters is very limited. Different values for the hyperparameters are defined on a grid and the network is trained for each combination. The network that has the lowest validation loss during training, is chosen as the optimal model for the Boltzmann distribution. If one of the best hyperparameters is at the edge of its range, the grid search is repeated with a more expanded range.

3.4 GENERATING SAMPLES

Once the network is trained, it can be used to generate new independent configurations $\mathbf{s}$ with corresponding normalised probabilities $q(\mathbf{s}; \theta)$ of the one-dimensional Ising model.

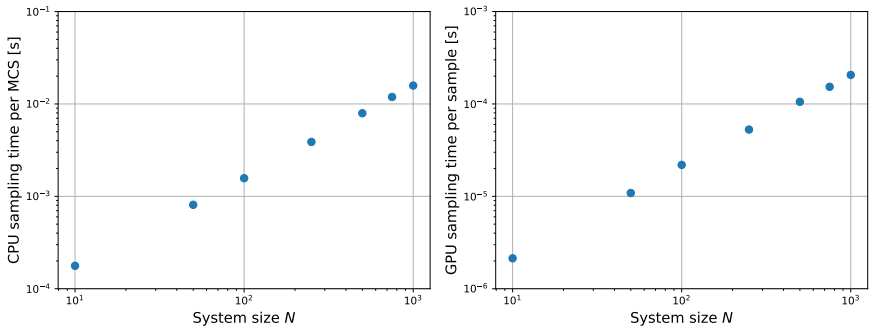

Figure 12: *Scaling of the time complexity, which is the average computational time per sampled configuration* $\mathbf{s}$*, with system size* N*. The MCMC method (left) is an average over* $n = 10^4$ *generated samples on an* Intel(R) Xeon(R) CPU @ 2.00GHz*. The RNN sampling (right) is an average of ten batches of* $n = 10^4$ *configurations generated at once on a* Tesla P4 GPU.

Often we are interested in the physics for large system sizes; in particular the thermodynamic limit $N \to \infty$. The MCMC algorithm, described in Appendix A.1, scales linearly with system size $\mathcal{O}(N)$, since each generated sample per Monte Carlo step per spin (MCS) requires an iteration over the different lattice sites. In the same way, the sampling procedure of the RNN requires an iteration over the sites (see Figure 7) and also scales as $\mathcal{O}(N)$.

However, there is an important difference between the two sampling processes. Markov chains are inherently sequential since the configuration x_i is needed to generate the configuration x_{i+1}. Therefore it is difficult to speed up the sampling process. Subsequent samples can also be highly correlated, meaning that not all generated samples are equally qualitative. Hence, there are many more iteration steps needed to collect the same number of qualitative samples than for sampling without correlations. The RNN sampling is also sequential but can be easily parallelised by generating batches of independent configurations. In particular, a GPU can speed up the simulations. For the implementations considered here, the RNN generates configurations much more efficiently than MCMC algorithms.

In Figure 12, the sampling time complexity of the MCMC and RNN is compared to each other for scaling system size. For MCMC we define this as the average CPU time per generated sample, where a configuration is sampled for each MCS. For the RNN we define this as the average GPU time per generated sample when sending a batch of configurations through the network. When the system size is increased by an order of magnitude, the sampling time is also approximately increased by an order of magnitude. The leading order of the scaling is linear and corresponds with the theoretical $\mathcal{O}(N)$. Unfortunately, it is difficult to quantitatively compare sampling since the two methods

are not using the same hardware. By optimising the RNN sampling process and using multiple GPUs, it is possible to sample configurations even more efficiently. The MCMC sampling can also be accelerated but typically requires more insight and effort [42].

Contrary to MCMC where the simulations directly generate samples, the RNN has to be properly trained first. Neural Networks are trained as an optimisation procedure which is typically computationally quite expensive. Therefore the majority of the effort goes into training the network; once it is properly trained, sampling is relatively easy and efficient.

3.5 PHYSICAL RESULTS

With the configurations, generated by the network, the physical quantities from Table 1 can be estimated. Since the RNN samples have normalised probabilities, it is possible to compute the free energy and the entropy of the system.

A naive approach is to take the samples and immediately estimate the observables using (13) and their error using (12). However, there is an important distinction to be made here. The samples are not drawn from the Boltzmann distribution $p(\mathbf{s})$, but from a distribution which is hopefully very similar $q(\mathbf{s};\theta)$. Directly using the generated samples results in estimating $\langle O(\mathbf{s})\rangle_{\mathbf{s}\sim q}$ instead of $\langle O(\mathbf{s})\rangle_{\mathbf{s}\sim p}$. Generally, the NN cannot exactly match the Boltzmann distribution and this discrepancy introduces a bias into the estimates. For the same reason, there is no clear method to obtain reliable error estimates for the estimates of the physical quantities [38].

Luckily, [38] introduces two techniques to obtain asymptotically unbiased estimates by utilising either importance sampling or MCMC sampling. Here, the asymptotically unbiased estimations and their errors are implemented using *Neural Importance Sampling (NIS)*, which estimates the physical observable $\langle O\rangle_{\mathbf{s}\sim p}$ as

$$\widehat{O}_n = \sum_{i=1}^{n} w_i O(\mathbf{s}_i) \qquad \text{with} \qquad \mathbf{s}_i \sim q(\mathbf{s};\theta), \tag{24}$$

where w_i are the normalised importance weights, obtained by normalising the importance weights $\widetilde{w}_i$:

$$w_i = \frac{\widetilde{w}_i}{\sum_{i=1}^{n} \widetilde{w}_i} \qquad \text{with} \qquad \widetilde{w}_i = \frac{\exp\left(-\beta E(\mathbf{s}_i)\right)}{q(\mathbf{s}_i;\theta)}. \tag{25}$$

Neural Importance Sampling also provides reliable estimates for the estimates of physical observables; the full details about NIS can be found in Appendix A.2.

In Figure 13 (left) the connected correlation function is depicted for a one-dimensional Ising system at low temperature. The NIS estimates

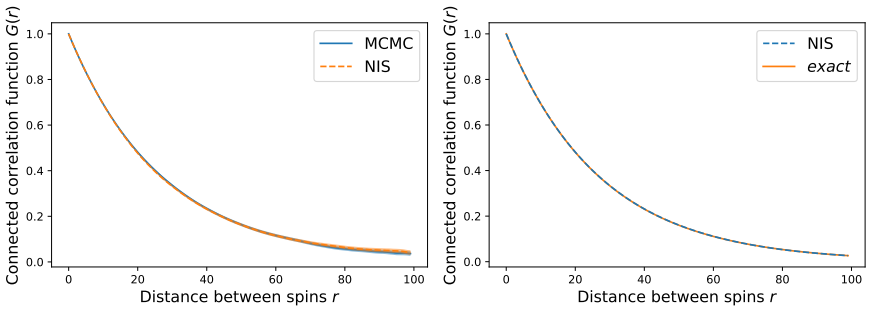

Figure 13: *The connected correlation function* $G(r) = \langle s_i s_{i+r} \rangle - \langle s_i \rangle \langle s_{i+r} \rangle$ *and its standard deviation (barely or not visible), estimated with* $n = 10^4$ *(left) and* $n = 10^6$ *(right) samples from the RNN using NIS. The* $G(r)$ *are compared with the MCMC ones using* $n = 10^4$ *samples (left) and exact ones for* $N \to \infty$ *(right). The one-dimensional Ising has* $N = 100$ *spins,* $\beta J = 2$ *and* $\beta H = 0$*. The optimised network has* $P_\theta = 31$ *number of parameters.*

are compared to the MCMC ones using the same number of samples. The network can describe the correlations in the system to an accuracy comparable to the MCMC data, which was used to train the model. Although the NIS and MCMC estimates are consistent with each other, the variance of the NIS estimates is noticeable higher at large distances r. Potential causes are: (1) the correlations $\langle s_i s_{i+r} \rangle$ are estimated by averaging over all spin pairs at a distance r and there are many more spin pairs at shorter than at larger distances or (2) some artefact of the recurrent behaviour of the network architecture. Figure 13 (right) shows a comparison of the NIS correlations with the analytical solution $(N \to \infty)$, but estimated with more samples. The estimations almost perfectly coincide with the analytical solution, even at large distances between spins. This suggests that the decreased accuracy and variance at large distances are caused by the limited number of averaged samples and not by the network.

Table 5 lists the estimated physical quantities for the system. Notice that the original data does not provide an estimation for the free energy and the entropy but the RNN does. For both the magnetisation and energy, the NIS estimate of the network coincides with the MC estimate from the data. For a system at high temperature (and small β), the free energy $\beta F = \beta E - S$ is dominated by the entropy term. The spins are randomly aligned and the system is disordered. On the other hand, at low temperatures, the free energy is dominated by the energy term. When $\beta J = 2$, the free energy is primarily dominated by the energy of the system and has a small entropy contribution.

The results from Figure 13 and Table 5 confirm that the RNN architecture can sample configurations from the Boltzmann distribution of the one-dimensional Ising system when it is trained on MCMC data.

PHYSICAL QUANTITY	MCMC	NIS
Magnetisation m	0.0065 ± 0.0063	0.0024 ± 0.0064
Energy βE	-190.8308 ± 0.0534	-190.9636 ± 0.0528
Free energy βF	✗	-200.4886 ± 0.0004
Entropy S	✗	9.5250 ± 0.0529

Table 5: *The relevant physical quantities of the system and their standard deviations, estimated with $n = 10^4$ generated samples of the RNN with NIS. The one-dimensional Ising has $N = 100$ spins, $\beta J = 2$ and $\beta H = 0$. The MCMC estimates with standard deviations, obtained by simulating $n = 10^4$ samples, are also shown.*

Figure 14 shows a visualisation of the generated configurations of the network. The different configurations are sorted according to the network probability $q(\mathbf{s};\theta)$ and their corresponding energy and free energy are computed. When the model is trained well, the network distribution approximates the Boltzmann one

$$\begin{aligned} q(\mathbf{s};\theta) &\approx p(\mathbf{s}) = \frac{1}{Z}\exp\left(-\beta E(\mathbf{s})\right) \\ \Leftrightarrow \ln q(\mathbf{s};\theta) &\approx -\ln Z - \beta E(\mathbf{s}). \end{aligned}$$

If the configurations $\mathbf{s}$ are sorted according to the probabilities $q(\mathbf{s};\theta)$, they are also sorted according to the energies $\beta E(\mathbf{s})$ since Z is a constant. For a system at low temperature, one expects the ground states $\mathbf{s} = (\uparrow\uparrow \ldots \uparrow)$ and $\mathbf{s} = (\downarrow\downarrow \ldots \downarrow)$ to dominate the energy spectrum of the generated configurations. This is exactly the case: the up and down chain are the configurations with highest probability $q(\mathbf{s};\theta) \approx 10^{-1}$. Since the temperature is finite ($\beta < \infty$), the network has to generate the first excited states. The first excited state of the one-dimensional Ising has one boundary in the spin chain, where a spin neighbours an opposite spin: $(\uparrow\uparrow\uparrow\downarrow\downarrow\downarrow)$ or $(\downarrow\downarrow\downarrow\uparrow\uparrow\uparrow)$. In Figure 14 (top) we see that the next states with the highest probability are precisely these first excited state with a domain wall of length one. The differences between the occupation probabilities of the first excited state (top figure) are much smaller than the difference between the probabilities of the first excited state and the second excited state (bottom figure), where the spin chains have two boundaries. Therefore, the network can separate the different energy states of the one-dimensional Ising which is the most important result.

First, notice that the network probability of these states is slightly different depending on where the boundary occurs (top figure). It looks like the network learns the first excited state according to a sequence: this is probably reminiscent of the sequential behaviour of the RNN architecture since it repeatedly applies the same RNN cell

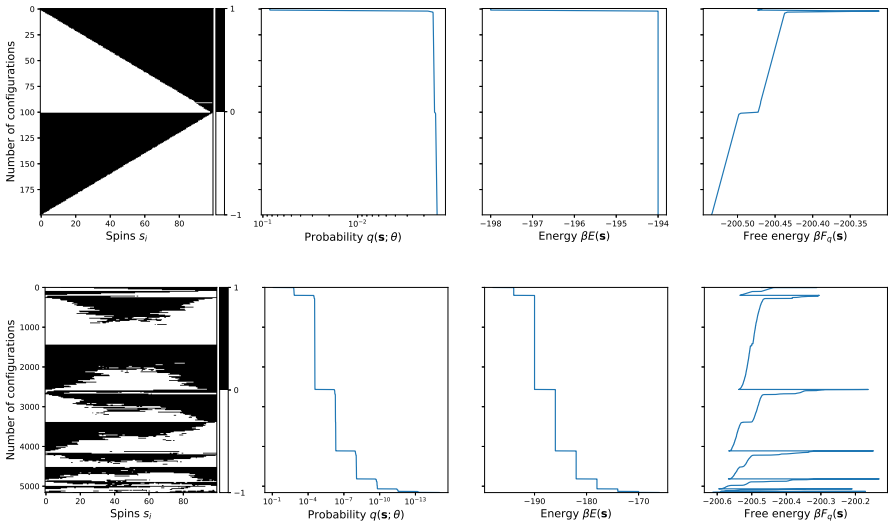

Figure 14: *A visualisation of the generated configurations* $\mathbf{s}$ *for a one-dimensional Ising with* $N = 100$, $\beta J = 2$ *and* $\beta H = 0$. *The figures represent the different spin chains that were generated by the* RNN, *sorted according to the network probability* $q(\mathbf{s};\theta)$. *The spins are visualised (left) as black* $(s = +1)$ *and white* $(s = -1)$. *The energies* $\beta E(\mathbf{s})$ *and the free energies* $\beta F(\mathbf{s})$ *are also depicted. The top figures represent the* 200 *configurations with the highest network probability, while the bottom figures represent all* 5161 *different configurations of the* $n = 10^4$ *that were generated. The system has* 2^{100} *different possible configurations.*

on the information flow. For example, the network first learns the configuration with the boundary between s_1 and s_2; then between s_2 and s_3 and continues until the boundary between s_{N-1} and s_N. From experimental simulations, it turns out that this sequential behaviour is typical for systems at low temperatures. In that case, a clear black-white pattern is visible in the visualisation of the configurations.

Second, since a hundred configurations with the highest probability are the flipped spin versions of the next hundred ones, it shows that the network does not fully learn the $\mathbb{Z}_2$ symmetry of the system. Therefore the distribution should make no distinction between a configuration $\mathbf{s}$ and $-\mathbf{s}$. We come back to this in chapter 4, where a simple symmetrisation scheme is introduced during training to fix this problem.

Finally, the bottom figure proves that the network can learn the discrete energy levels of the one-dimensional Ising chain: each step in the energy profile corresponds to an energy level. The step with the highest probability corresponds to the ground state and the steps with lower probabilities to the higher energy states.

To illustrate the power and usefulness of NIS, we test how much MCMC training data is needed to achieve reasonable results. We train different networks on random subsets of the data. The results for the estimated physical quantities can be seen in Figure 15.

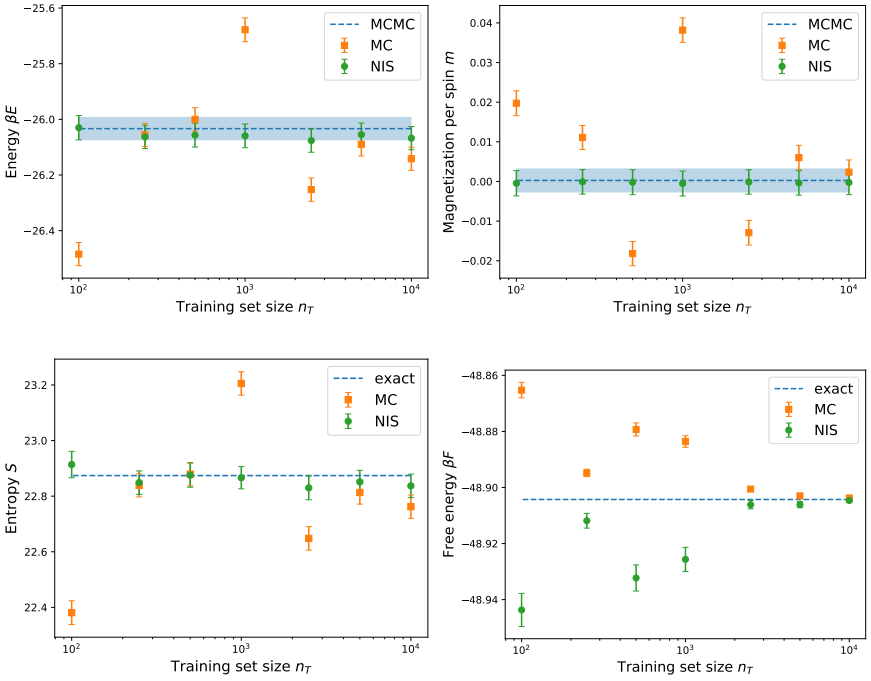

Figure 15: *A comparison between the naive MC and NIS estimates of physical observables using* $n = 10^4$ *generated spin configurations from networks trained on a data set with* n_T *samples. The magnetisation and the energy are compared to MCMC estimates with* $n = 10^4$ *samples. The entropy and free energy are compared to the analytical solution.*

The naive approach, without NIS, has difficulties obtaining reliable estimations, sometimes over -or underestimating the MCMC estimate. On the contrary, NIS provides reliable results for small training sets consistent with MCMC. For larger training sets, the naive estimations typically improve and approach the MCMC estimates. It is definitely worth implementing NIS to estimate the physical quantities, especially for smaller training sets.

For the entropy, only estimations of the network are available. Therefore we compare with the analytical solution. The NIS estimates are consistent with the exact solution while the MC estimates display large variations. The network is not fully captured and introduces a bias into the estimations, especially for small data sets.

The energy and entropy estimations can be combined to a free energy estimation via $\beta\widehat{F} = \beta\widehat{E} - \widehat{S}$. Notice the small scale on the vertical axis of the free energy graph. Apparently, both MC and NIS estimations are capable to predict the free energy in a relatively small range of values. This coincides with the *zero variance principle*, which is discussed in Appendix C.1 and chapter 4. The small variance for the free energy estimations probably also leads to the very similar behaviour of the estimated energy and entropy, which are estimated so that $\beta\widehat{F} = \beta\widehat{E} - \widehat{S}$ is in a very small range around the true value. The apparent symmetry between the MC and NIS estimations for the free energy is not immediately clear. One can see, however, that the predictions and errors converge to the analytical solution for larger training sets. This suggests that to get higher accuracy in the estimations, it is beneficial to train with a larger data set.

3.6 CONCLUSION

By training the RNN architecture on a pre-generated MCMC data set and minimising the NLL, the model is successful at capturing the physics of the one-dimensional Ising system. The correlations and physical quantities are consistent with MCMC estimations.

The optimised networks need only very few parameters to learn the distribution and capture the physics: a hidden dimension $h_d \approx 1-5$ and $n_l \approx 1-2$ layers are sufficient. This corresponds to a total of $P_\theta \approx 19-149$ trainable parameters. When comparing this to Figure 9, where overfitting only occurs for a very large number of parameters, there is no need for regularisation during the network training.

Numerical experiments show that most hyperparameters of the network do not have a substantial impact on the results: both during training and the final physical results. The network needs sufficient complexity to describe the system. Once it has reached this complexity, adding more parameters only introduces minor differences in the performance of the network. This indicates that most tuned networks find a set of parameters θ that produce accurate results.

The RNN architecture is a one-dimensional construction and respects the locality of the Ising Hamiltonian, which makes it an ideal candidate for describing the one-dimensional Ising system. Therefore, despite achieving good results for this particular system, it cannot be expected that this RNN architecture can describe more complicated systems.

If the configurations of the system are readily available, for example from a physical experiment [4, 41], then this method is a simple and easy way to train NNs: use the data as input and minimise the NLL. Once the network is trained, configurations can be independently and efficiently sampled from the network. On the other hand, if the data is not readily available as is the case here, then it can get expensive to construct qualitative data to train the network. Here, the samples were generated using the MCMC method but the ultimate goal is to avoid the disadvantages which Markov chains introduce. However, it is possible to train the network on data produced by itself and avoid the complications with gathering training data [55].

4 VARIATIONAL LEARNING

To avoid the complications connected with the data process, [55] proposes a variational ML method where the network is trained on configurations generated by the network itself. In that case, there is no need to put any effort into collecting qualitative data. The complexity of solving the problem then comes down to optimising and training the network, which can remain very challenging.

Consider the one-dimensional Ising model with open boundary conditions from chapter 1. Contrary to chapter 3, the network is trained using an alternative training method that does not require pre-generated configurations of the system. We are using the same RNN architecture from section 3.2.

4.1 TRAINING THE NETWORK

The optimal parameters θ that minimise the dissimilarity between the network distribution $q(\mathbf{s};\theta)$ and the Boltzmann distribution $p(\mathbf{s})$, can be found starting from the *reversed Kullback-Leibler divergence*

$$D_{KL}(q \,\|\, p) = \sum_{\mathbf{s}} q(\mathbf{s};\theta) \ln\left(\frac{q(\mathbf{s};\theta)}{p(\mathbf{s})}\right) = \beta F_q - \beta F, \tag{26}$$

where the normalisation of the variational distribution $\sum_{\mathbf{s}} q(\mathbf{s};\theta) = 1$ is used. The reversed KL divergence can be written in terms of $\beta F_q \coloneqq \sum_{\mathbf{s}} q(\mathbf{s};\theta)\,(\beta E(\mathbf{s}) - \ln q(\mathbf{s};\theta))$, which is the variational free energy according to the variational distribution $q(\mathbf{s};\theta)$, and $\beta F = -\ln Z$, which is the free energy of the system. Due to the Jensen inequality, βF_q is an upper bound for the free energy βF.

There is an important distinction between the reversed (26) and the standard (21) KL divergence. The standard version corresponds to distribution estimation: given configurations drawn from the Boltzmann distribution $\mathbf{s}$, find the parameters θ so that $q(\mathbf{s};\theta)$ approximates $p(\mathbf{s})$. For the reversed version, there is no need for configurations sampled according to the Boltzmann distribution. Minimising (26) is equivalent to minimising the variational free energy, which is an expectation value over samples from the variational distribution

$$\beta F_q = \langle \beta E(\mathbf{s}) - \ln q(\mathbf{s};\theta) \rangle_{\mathbf{s}\sim q}\,, \tag{27}$$

with $E(\mathbf{s})$ the energy of the one-dimensional Ising chain. The variational free energy is often used as an estimate of the free energy: if

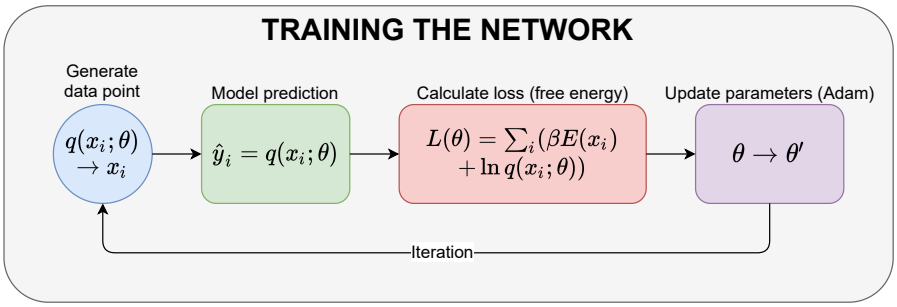

Figure 16: *A flowchart of the training process: a batch of spin configurations x_i is generated by the network and then forwarded through the network to predict the probabilities $\hat{y}_i$. The x_i are used to calculate the free energy loss $L(\theta)$. Finally, a back-propagation through the network provides the gradients of the loss and are used by the Adam optimiser to update the parameters $\theta \to \theta'$. This procedure is repeated until the training has iterated a fixed number of times. We choose the best model based on the parameters that lead to the minimal validation loss.*

the two distributions are exactly the same $p = q$, then $\beta F_q = \beta F$. In practice, the expectation from (27) is approximated by an MC estimate. The actual loss used during training is

$$L(\theta) = \frac{1}{n_b} \sum_{i=1}^{n_b} \left(\beta E(\mathbf{s}_i) - \ln q(\mathbf{s}_i;\theta)\right), \tag{28}$$

with n_b the number of generated configurations in one iteration. For the practical implementation of the training approach, we refer to Appendix C.1. The network is trained by generating configurations of the system, minimising an estimation of the variational free energy and using the Adam optimiser for updating the parameters. The training flow is visualised in Figure 16.

First, we notice that the training is very similar to chapter 3: instead of selecting a batch of samples to train the model, a batch of samples is generated. Instead of calculating the average loss after one iteration through the entire data set, the average loss is calculated after a fixed number of training steps N_T.

Second, in chapter 3 it was shown from numerical simulations that the hyperparameters have a small impact on the training so that most networks produce approximately the same results. In [55] it is mentioned that this behaviour also occurs for a variational training approach. Hence, the grid search from section 3.3 is omitted, thereby reducing the total training time. After all, the training of the network is computationally more expensive since the configurations are generated during training. The hyperparameters are fixed based on grid searches from chapter 3.

Third, since new configurations are generated for each iteration, the model is very unlikely to overfit to the data. Validating the net-

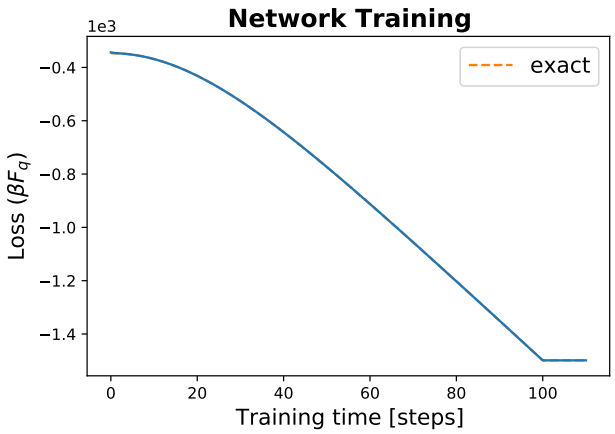

Figure 17: *The estimated free energy loss (28) with standard deviation (not visible) during training of a network with* $P_\theta = 147$ *parameters. The one-dimensional Ising has* $N = 500$ *spins,* $\beta J = 3$ *and* $\beta H = 0$*. The temperature is annealed from* $\beta_0^* J = 0$ *to* $\beta J = 3$ *in* $N_\Lambda = 100$ *steps. At every temperature, the network is trained for* $N_T = 10$ *parameter updates with batch size* $n_b = 10^3$*. After annealing, the network is further trained for ten steps with* $N_T = 10$*. The free energy loss is compared to the analytical solution (barely visible under the loss).*

work after every epoch results in very similar training and validation losses. This is definitely not always the case, as can be seen in Figure 9. Therefore, much time can be saved by omitting the validation of the model during training.

Finally, it is possible that the network is not able to capture all modes (pure states) of the distribution $p(\mathbf{s})$ of the system [55]. The pure states of the one-dimensional Ising are the up and down chain. For low temperatures, we use temperature annealing to prevent mode collapse by starting from a high temperature β_0^* and carefully anneal to the system temperature β. We choose a linear scheme where each annealing step is equal in size. At each annealed temperature, the parameters of the network approximately describe the equilibrium system at β^*. These parameters are then reused for the next equilibrium at $\beta^* + \Delta\beta^*$. The parameters are updated from one equilibrium to the next to prevent mode collapse [22]. After annealing, the network is further trained at the system temperature. For the one-dimensional Ising, mode collapse is only a concern at very low temperatures ($T_c = 0$). The two-dimensional Ising, for example, already has two modes at finite temperatures ($T_c > 0$). Other systems such as spin glasses can have many pure states [53]. More details about mode collapse and temperature annealing can be found in Appendix C.1.

Combining all these aspects of the training approach, we train networks for learning the Boltzmann distribution of the one-dimensional Ising system. An example can be seen in Figure 17. The estimated variance of the variational free energy remains small during anneal-

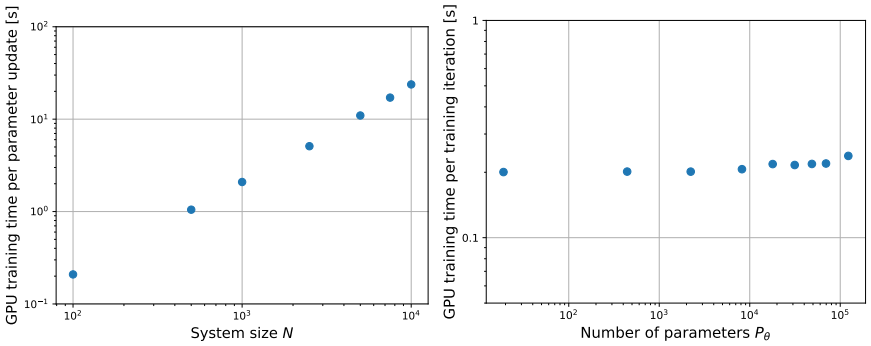

Figure 18: *Scaling of the time complexity, the average computational time per training iteration, with system size* N *(left) and number of parameters* P_θ *(right). The number of parameters is increased by increasing the hidden dimension* $(h_d = 1, \ldots, 200)$*. The networks are trained on a* Tesla K80 GPU.

ing. Hence if the annealing prevents mode collapse, the network finds the correct distribution including both the up and down chain.

One training step contains two separate iterations over the system size N: one for calculating the energy in the loss (28) and one during a forward pass through the network (see Figure 6). Therefore the average training time scales as $\mathcal{O}(N)$, similar to the sampling time. Figure 18 (right) shows the time complexity of the training process, here defined as the average computational time per training iteration, for scaling system size. If the system size is increased by an order of magnitude, then the average training time also approximately increases by an order of magnitude. The empirical data thus agrees with the theoretical linear scaling of $\mathcal{O}(N)$.

Figure 18 (left) depicts the scaling of the computational complexity with the number of parameters of the network P_θ. We vary the hidden dimension because the number of layers is typically small when training RNNs [19]. Deep Learning libraries are designed and optimised to work with millions of parameters [39]. If the network is trained on a GPU, there is no significant computational cost by training with a larger hidden dimension.

4.2 PHYSICAL RESULTS

Once the network is properly trained, the RNN can be used to generate configurations of the system, which in turn are used to estimate physical quantities of the system under study.

In Table 6 the physical quantities are estimated and compared to the MCMC estimations from Table 5. The results for the energy and the magnetisation are consistent with the MCMC estimations. This proves that the alternative training approach can learn the correct distribution with the extra advantage that no efforts are needed for data col-

PHYSICAL QUANTITY	MCMC	NIS
Magnetisation m	0.0065 ± 0.0064	0.0004 ± 0.0063
Energy βE	-190.8308 ± 0.0534	-190.8193 ± 0.0534
Free energy βF	✗	-200.4900 ± 0.0001
Entropy S	✗	9.6707 ± 0.0534

Table 6: *The relevant physical quantities of the one-dimensional Ising and their standard deviations, estimated with $n = 10^4$ samples from the RNN with NIS. The system has $N = 100$ spins, $\beta J = 2$ and $\beta H = 0$. The MCMC estimates with standard deviations, obtained by simulating $n = 10^4$ samples, are also shown.*

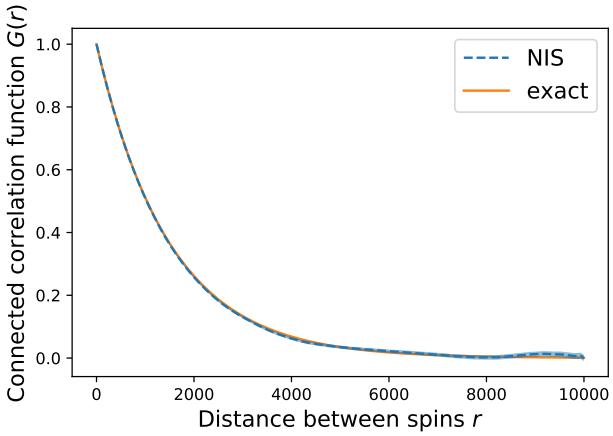

Figure 19: *The connected correlation function $G(r)$ and its standard deviation (barely visible), estimated with $n = 10^4$ samples from the RNN using NIS. The one-dimensional Ising has $N = 10^4$ spins, $\beta J = 4$ and $\beta H = 0$. The exact correlations $(N \to \infty)$ are also depicted. During training, the temperature is annealed for $N_A = 10$ steps, each with $N_T = 10$ parameter updates. After annealing, the system is further trained for ten steps each with $N_T = 10$.*

lection or generation. The estimated free energy has a small variance in comparison to the other estimations. This coincides with the zero variance principle (see Appendix C.1): if there is no mode collapse, a small variance of the free energy suggests that $q(\mathbf{s};\theta) \approx p(\mathbf{s})$. The energy of the generated configurations is distributed according to a distribution with a larger variance since the network should generate configurations with lower energies (ground state) and higher energies (excited states). For the entropy it holds that $S = \beta E - \beta F = \beta + \ln Z$, where the partition function Z is a constant normalisation. If the free energy has a low variance, the entropy S has a similar distribution as the energy βE. In particular, they have a similar variance.

Figure 19 shows the connected correlation function $G(r)$ for a system at low temperature. The larger variance at large distances r is

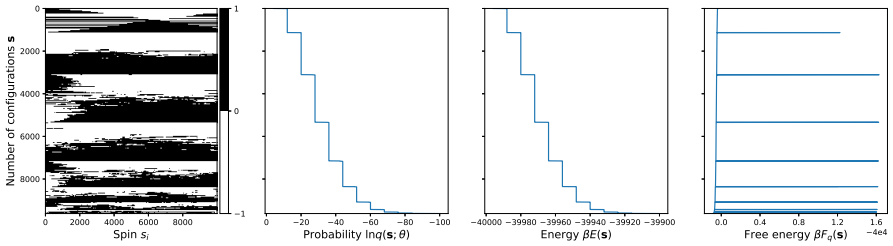

Figure 20: *A visualisation of the generated configurations* $\mathbf{s}$ *for a one-dimensional Ising with* $N = 10^4$, $\beta J = 4$ *and* $\beta H = 0$. *The figures represent the different spin chains that were generated by the* RNN, *sorted according to the network probability* $q(\mathbf{s};\theta)$. *The spins are visualised (left) as black* $(s = +1)$ *and white* $(s = -1)$. *The energies* $\beta E(\mathbf{s})$ *and the free energies* $\beta F(\mathbf{s})$ *are also depicted. The figures represent 9620 different configurations of the* $n = 10^4$ *that were generated. The system has* 2^N *possible configurations.*

caused by the limited number of spin pairs (see section 3.5). The generative model can describe the correct correlations in the system. It is possible to train networks for larger system sizes on a single GPU at a relatively small computational cost. Optimising the training algorithm or using multiple GPUs allows the network to learn distributions for very large systems, which is quite difficult using traditional MCMC algorithms. Once the network is properly trained, configurations can be efficiently and independently sampled in the same way as section 3.4. Therefore the computational cost is dominated by training the network.

Figure 20 shows a visualisation of the generated configurations of the network, similar to Figure 14. The network can generate the correct spin configurations: the ground states have the highest probability and dominate the spectrum at low temperature, while the states with high excitation energy have a low Boltzmann probability.

We can exploit the relation between the locality of the system and the RNN cell (see section 3.2). The cell should describe the physics of a single site of the system and because of the locality of the one-dimensional Ising, this information should be the same for every site in the chain. In particular, the physical quantities only depend on the system parameters βJ and βH and scale extrinsically with the system size N. Therefore, the architecture allows an RNN cell to be trained for a system size N_1 and be used to generate configurations for N_2. The result can be seen in Figure 21. The generated configurations can describe the correlations for N_2. However, remember that this only works because of the local physics of the Ising model and the full parameter sharing of the RNN. Therefore this cannot be considered as a general technique.

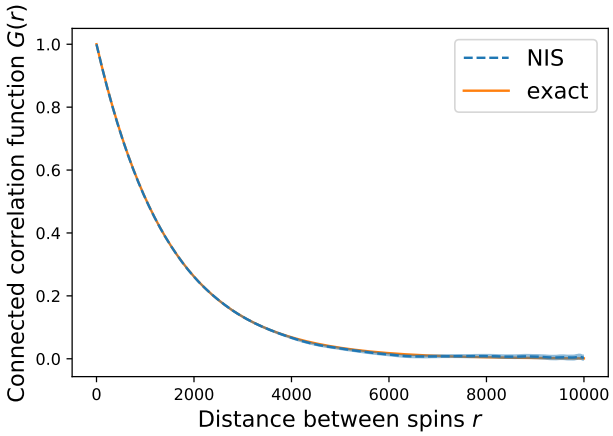

Figure 21: *The connected correlation function* $G(r)$ *and its standard deviation (barely visible), obtained with* $n = 10^4$ *generated samples of the* RNN *using* NIS*. The network is trained for a one-dimensional Ising system with* $N_1 = 10$ *spins,* $\beta J = 4$ *and* $\beta H = 0$*, but the configurations are generated for* $N_2 = 10^4$ *spins. The analytical solution in the thermodynamic limit* $(N \to \infty)$ *is also depicted.*

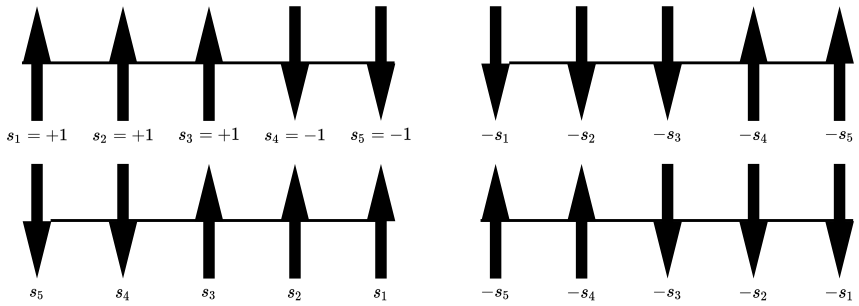

Figure 22: *A visualisation of the symmetries of an open one-dimensional Ising chain with* $N = 5$ *spins. The figure shows an example spin chain* $\mathbf{s}$ *(top left) and its symmetry invariant configurations: flipped* $-\mathbf{s}$ *(top right), reflected* $\mathbf{s}_R$ *(bottom left) and flipped and reflected* $-\mathbf{s}_R$ *(bottom right).*

4.3 IMPOSING SYMMETRY

When introducing a variational ansatz for the Boltzmann distribution, the symmetries of the system can be incorporated into the variational distribution $q(\mathbf{s}; \theta)$. This imposes the variational distribution to exhibit the same symmetries as the Boltzmann one [29].

Consider the Hamiltonian of the one-dimensional Ising with vanishing field $H = 0$ and open boundary conditions

$$E = -J \sum_{i=1}^{N-1} s_i s_{i+1},$$

which has the following two symmetries that are visualised in Figure 22 [37]:

- $\mathbb{Z}_2$ *symmetry*: the Hamiltonian is invariant under flipping of all the spins $s_i \to -s_i$ with $i = 1, \dots, N$. The system makes no distinction between a configuration $\mathbf{s}$ and $-\mathbf{s}$ so that $E(\mathbf{s}) = E(-\mathbf{s})$ and, in particular, $p(\mathbf{s}) = p(-\mathbf{s})$. This symmetry can be built into the variational distribution by setting the probability [29, 55]

$$q_{\text{sym}}(\mathbf{s}; \theta) = \frac{1}{2} \left(q(\mathbf{s}; \theta) + q(-\mathbf{s}; \theta) \right), \tag{29}$$

  where q_{sym} is the symmetrised variational distribution.

- *Reflection symmetry*: the Hamiltonian is also invariant under reflection of the spins between the boundaries $s_i \to s_{N-i+1}$ with $i = 1, \dots, N$. Similarly, the system should make no distinction between a configuration $\mathbf{s}$ and its reflection $\mathbf{s}_R$ so that $p(\mathbf{s}) = p(\mathbf{s}_R)$. The symmetrised variational distribution is then

$$q_{\text{sym}}(\mathbf{s}; \theta) = \frac{1}{2} \left(q(\mathbf{s}; \theta) + q(\mathbf{s}_R; \theta) \right). \tag{30}$$

The two symmetries (29) and (30) can be combined into a symmetrised variational distribution by averaging the probabilities over the symmetry operations

$$q_{\text{sym}}(\mathbf{s}; \theta) = \frac{1}{4} \left(q(\mathbf{s}; \theta) + q(-\mathbf{s}; \theta) + q(\mathbf{s}_R; \theta) + q(-\mathbf{s}_R; \theta) \right). \tag{31}$$

Since the four probabilities are properly normalised, $q_{\text{sym}}(\mathbf{s}; \theta)$ is also normalised. Not only does the q_{sym} respect the symmetries of the system, there are also fewer configuration-probability couples that have to be learnt by the network. Despite the computational overhead by calculating three probabilities instead of one, the training of the network should be easier and, eventually, training can be computationally less expensive than for $q(\mathbf{s}; \theta)$.

In addition, one can prove that the true loss of the symmetrised model is equal to or lower than the non-symmetrised version [29] (see Appendix C.2 for a derivation):

$$\beta F_q \geqslant \beta F_{q_{\text{sym}}}.$$

This suggests that the network can find a lower variational free energy $\beta F_{q_{\text{sym}}}$ during training. It follows, per the assumption of the

	CONFIGURATION $\mathbf{s}$	$\ln q_{sym}(\mathbf{s};\theta)$	$\ln q(\mathbf{s};\theta)$
1	$(\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow)$	-1.8380	-1.8364
2	$(\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow)$	-1.8380	-1.8393
3	$(\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\uparrow)$	-3.8124	-3.8369
4	$(\downarrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow)$	-3.8124	-3.8286
5	$(\uparrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow)$	-3.8124	-3.7985
6	$(\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\downarrow)$	-3.8124	-3.8379
7	$(\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\downarrow\downarrow\downarrow)$	-3.8361	-3.8323
8	$(\downarrow\downarrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow\uparrow)$	-3.8361	-3.8300
9	$(\uparrow\uparrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow)$	-3.8361	-3.8536
10	$(\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\downarrow\uparrow\uparrow)$	-3.8361	-3.8353

Table 7: *Ten generated configurations* $\mathbf{s}$ *from the* RNN *with the highest probability* $q_{sym}(\mathbf{s};\theta)$*, for a one-dimensional Ising with* $N = 10$*,* $\beta J = 1$ *and* $\beta H = 0$*. The* q_{sym} *are compared to the probabilities* q*, obtained without imposing symmetry.*

KL divergence, that the learnt network distribution is closer to the Boltzmann distribution. As a result, a symmetrised variational distribution could, via (31), make learning easier since the symmetry reduces the complexity of the problem.

By training the model with the symmetrised probabilities in the loss (28), the network is learning the distribution q_{sym} instead of q. Therefore, the symmetrised probabilities are the only ones that matter, meaning that the sampling should also be done according to q_{sym}. The sampling procedure is symmetrised as follows: a batch of configurations is generated by the RNN. Then each configuration $\mathbf{s}$ is flipped with $1/2$ probability $\mathbf{s} \to \mathbf{s}'$ ($\mathbb{Z}_2$ symmetry) after which it is reversed with $1/2$ probability $\mathbf{s}' \to \mathbf{s}''$ (reflection symmetry). As a result, when $\mathbf{s}$ is generated, one of the four symmetrised configurations $(\mathbf{s}, -\mathbf{s}, \mathbf{s}_R$ or $-\mathbf{s}_R)$ is sampled with $1/4$ probability.

Table 7 shows the effect of symmetrising the variational ansatz. The symmetrised variational ansatz gives an equal probability to configurations that are invariant under the two symmetry operations: $\mathbf{s}$, $-\mathbf{s}$, $\mathbf{s}_R$ and $-\mathbf{s}_R$. There is still a small difference in probability between non-symmetry configurations (but within the same energy level). For example, the configurations $3-6$ have a slightly higher probability than the configurations $7-10$. However, the differences in probability between the different energy levels are much larger. For example, compare the configurations $1-2$ with $3-8$. Without imposing symmetry, the network gives a different probability to configurations that are invariant under the symmetry operations.

By imposing symmetries there are fewer probabilities to be learnt by the network. However, in practice, there is no noticeable difference

between network training with and without imposing symmetry. For the one-dimensional Ising model, in particular, there is no significant benefit by imposing symmetry.

4.4 CONCLUSION

The variational training minimises the variational free energy βF_q of configurations generated by the network. In combination with the RNN architecture, it captures the correct physics of the one-dimensional Ising. For this system, in particular, symmetrising the variational ansatz and sampling does not add a significant benefit to the training or estimations.

The variational approach is very similar to the data approach in many aspects concerning the training and physical results. For example, a total number of $P_\theta \approx 19 - 149$ trainable parameters is sufficient to learn the correct distribution. Because new configurations are generated at each parameter update, it is even more unlikely for the model to overfit to the configurations when compared to the data method. This suggests that overfitting and regularisation are certainly not of concern.

In the data approach, one needs an existing set of configurations that describe the Boltzmann distribution so that the network can estimate the density. The underlying distribution of the data must be similar to the Boltzmann one since one cannot expect the network to estimate the Boltzmann density if it is not encoded in the data. Neural Importance Sampling can help to fix the discrepancies between the learnt distribution and the Boltzmann one but fails if they are not similar enough. In any case, the data set needs to be carefully constructed with configurations from the Boltzmann distribution. The gathering of qualitative data is an extra step in the analysis of physical systems, which is completely omitted in the variational approach. The latter requires only a Hamiltonian of the system.

In the variational approach, the configurations are generated by the network during training, which is more expensive than iterating through existing data. Nonetheless, many aspects of ML and DL can be used to optimise and speed up the training of networks, including parallelisation and GPUs. In addition, autoregressive RNNs allow trained models to be iteratively reused for larger system sizes. The parameters of the RNN of a smaller system are used as initial parameters for a larger system, for which the model is retrained. This method can efficiently scale to very large system sizes [43].

In chapter 3 and chapter 4 we have only considered an architecture with a single RNN cell and fully connected layer, which is sufficient for the one-dimensional Ising. This is not necessarily the case for other dimensions or systems. More challenging systems likely require more intricate architectures [21]. In chapter 5 we consider lattice systems

with more difficult interactions between the spins and use different RNN architectures.

5

SPIN GLASSES

Consider the spin glass system described in section 1.3 with Hamiltonian $E = -\sum_{i<j} J_{ij} s_i s_j$. Similar to chapter 4, we want to minimise the variational free energy βF_q in combination with an RNN architecture to learn the Boltzmann distribution. However, there are three additional complications compared to the one-dimensional Ising with Hamiltonian $E = -J \sum_i s_i s_{i+1}$.

1. *Site-dependent interactions*: the interactions between the spins J_{ij} differ per site. While local interactions in spin glasses are very important, the transfer matrix is not the same for every site. A model has to learn different physics at every site, which increases the difficulty of learning.

2. *Long-range interactions*: a spin not only interacts with its nearest neighbours but also with spins at a longer distance. These interactions introduce long-distance dependencies between spins which are more difficult to learn for an RNN with full parameter sharing. This is similar to remembering long-term dependencies between inputs for an RNN (see Table 3). Capturing long-term dependencies with an RNN is a challenging problem in ML [14, 19].

3. *Random interactions*: the interactions are random and can be ferromagnetic ($J_{ij} > 0$) or antiferromagnetic ($J_{ij} < 0$). The system exhibits more challenging physics such as frustration and the existence of many metastable states [35].

The physics of the one-dimensional Ising, with its local ferromagnetic interactions, can be captured by applying the same RNN cell and fully connected layer Σ to each input s_i (chapter 3 and chapter 4). A one-dimensional spin glass is much more complicated, which suggests that a single RNN cell and fully connected layer are not sufficient to capture the physics. We investigate whether the RNN is suited to learn the distribution of the spin glass. If this turns out not to be the case, we consider different architectures so that the RNN can capture the Boltzmann distribution.

5.1 RANDOM ISING CHAIN

We first investigate the impact of site-dependent interactions by considering an open Ising chain with N spins and classical Hamiltonian

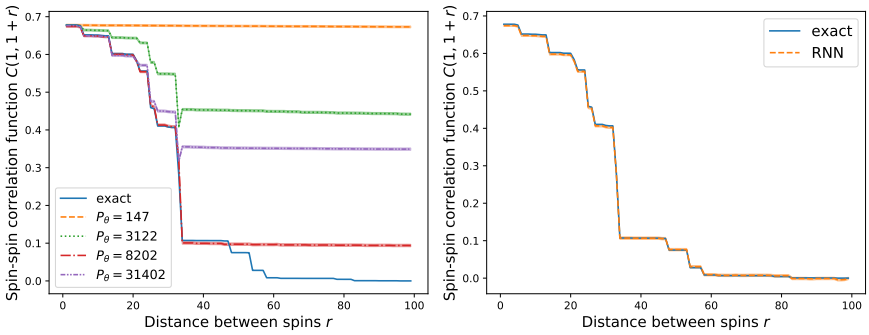

Figure 23: *The spin-spin correlation function* $C(i, i+r) = \langle s_i s_{i+r} \rangle$ *with* $i = 1$ *for a random Ising chain with* $N = 100$ *and* $\beta = 10$. *Left: three networks are trained using a single* RNN *cell and fully connected layer (see Figure 6) with increasing number of parameters* P_θ. *Right: a network is trained using a single* RNN *cell but a different fully connected layer for every site (see Figure 24) with* $P_\theta = 1335$ *parameters. The* $C(1, 1+r)$ *are estimated with* $n = 10^5$ *samples and compared to the analytical solution for finite systems with open boundary conditions [20].*

$$E = -\sum_{i=1}^{N-1} J_{i,i+1} s_i s_{i+1}.$$

The interactions are randomly distributed according to a uniform distribution between zero and one:

$$J_{i,i+1} \sim \mathcal{U}([0,1[) \qquad \text{with} \qquad \mathcal{U}([a,b[) := \begin{cases} \frac{1}{b-a} & a \leqslant x < b \\ 0 & x < a \text{ or } x \geqslant b \end{cases}.$$

The interactions are ferromagnetic and restricted to neighbouring spins, similar to the Ising model, but the $J_{i,i+1}$ are site-dependent. The ground states are the spin up and down chain. The system is characterised by a range of transfer matrices that are site-dependent. Hence, it is more difficult for a network to learn the distribution. This can be seen in Figure 23 (left), where an RNN is trained with a small number of parameters. The network does not fully describe the correlations in the system, while it can capture the physics of the standard Ising chain (see Figure 19). The expressiveness of the NN can be increased by increasing the number of parameters P_θ. Until $P_\theta = 8202$, a more expressive network better captures the correlations at small inter-spin distances. One expects, that if P_θ is large enough, the network can capture the correlations at any distance. This is not the case, however, as can be seen for $P_\theta = 31402$. The training of the network is harder since the minimal free energy loss seems to get stuck at $\beta F_{q,\min} = -523.69 \pm 0.35$. The larger standard deviation suggests that

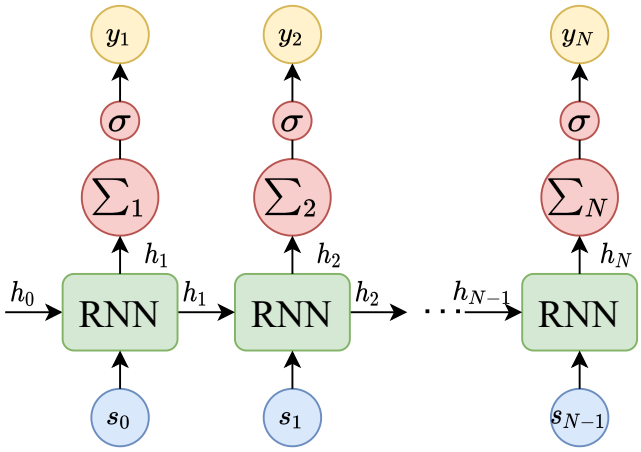

Figure 24: *A graphical representation of the RNN architecture, based on [21]. The network is initialised by the artificial spin s_0 and hidden vector $\mathbf{h}_0$. They are sent through an RNN cell (green box), fully connected layer Σ_i and softmax activation σ to provide the output probabilities y_i. Contrary to Figure 6, the architecture has a different fully connected layer for each input s_i.*

the network distribution does not approximate the Boltzmann one well. Getting stuck at a high-cost minimum has many potential causes such as local minima or vanishing gradients [19].

However, there exists a more elegant solution by considering an RNN architecture with a single cell and fully connected layers Σ_i for each site in the chain (see Figure 24). The position of each spin s_i is thus implicitly encoded in the parameters of the Σ_i. An intuitive interpretation is that the local interactions $J_{i,i+1}$ are represented by the passing of a hidden vector $\mathbf{h}_i$ through the chain and the Σ_i encode the different values of $J_{i,i+1}$. The combination of the two encodes the conditional probabilities $q_\theta(s_i \mid \mathbf{s}_{<i})$. When the parameters are shared, the values of the $J_{i,i+1}$ are encoded in the parameters of the single cell and Σ, which are shared over the spins. This explains why more parameters are needed to capture the physics. The estimated spin-spin correlations are shown in Figure 23 (right). With only a limited number of parameters ($h_d = 5$), the network can correctly estimate the correlations. Training is easier since the network quickly finds a lower free energy loss with a smaller standard deviation: $\beta F_{q,\min} = -526.69 \pm 0.04$. By assuming different fully connected layers, it is possible to keep the hidden dimension small. In practice, one can train networks for a large system size with few training steps. Similarly, one can build architectures with different RNN cells that do not share parameters or an RNN where the cells and the fully connected layers do not share parameters. Here, it suffices to forego parameter sharing on the fully connected layers and keep sharing the

weights for the cells. We conclude that it can be very beneficial to adapt the network architecture to the physical system under study.

5.2 LONG-RANGE ISING CHAIN

Next, we want to investigate long-range interactions for an open one-dimensional chain with N spins. Consider another classical Hamiltonian, where the range of the interactions is controlled by the parameter σ:

$$E = -\sum_{i<j}^{N} J_{ij} s_i s_j \qquad \text{with} \qquad J_{ij} = \frac{J}{r_{ij}^{2\sigma}}.$$

The interactions are ferromagnetic ($J > 0$) and decay as a power-law of the distance between the spins $r_{ij} = |i - j|$. For $\sigma \to \infty$, the system corresponds with the nearest neighbour Ising and for $\sigma = 0$ with the infinite-range Ising [6]. Notice that this is the same Hamiltonian as the spin glass one from section 1.3, but here the interactions are solely ferromagnetic and not random. Lattice models with power-law interactions have been extensively studied: while for $\sigma > 1$ there is no phase transition at finite temperatures ($T_c = 0$), for $\frac{1}{2} < \sigma < 1$ the system exhibits long-range order at finite temperatures ($T_c > 0$) [16]. The phase transition for $\frac{1}{2} < \sigma < \frac{3}{4}$ is characterised by mean-field exponents [44]. The critical exponents for $\frac{3}{4} < \sigma < 1$ are non-trivial and σ-dependent [36, 50]. When $\sigma = 1$, the system has a one-dimensional Kosterlitz-Thouless transition [58].

Consider the mean-field region ($\frac{1}{2} < \sigma < \frac{3}{4}$) with long-range order. In Figure 25 (left) different networks are trained with a single cell and Σ but an increasing number of parameters P_θ. The system exhibits long-range order at finite temperature. However, increasing the number of parameters has no significant effect on the estimated correlations. When compared to the MCMC estimate, none of the networks can correctly describe the correlations at large distances. This could be attributed to: (1) the RNN is not able to correctly capture the different interactions J_{ij}, similar to section 5.1 or (2) the RNN has difficulties capturing long-range dependencies in the system. The latter is a well-known challenge for RNNs in ML [14, 19].

When using a network with a single cell and multiple Σ_i, the correlations are consistent with MCMC, even at large inter-spin distances. This suggests that, at least for $N = 100$ and power-law interactions, long-range dependencies are not the most important challenge. This could be different for fully connected systems or systems with very long-range interactions.

We also mention that the energy is calculated with all $N(N-1)/2$ interactions. Hence for large system sizes N, calculating the energy can run into memory problems and/or very long simulation times.

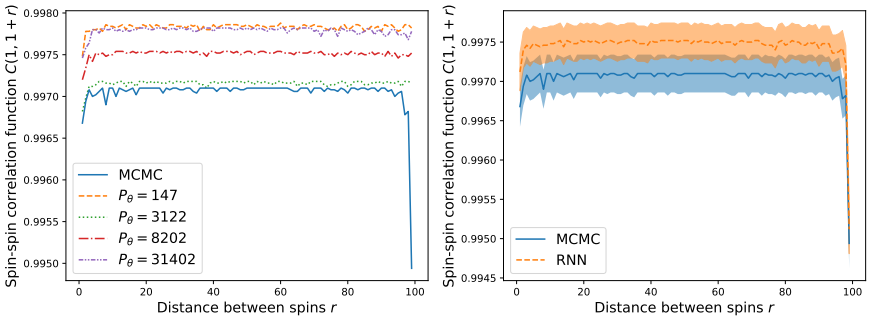

Figure 25: *The spin-spin correlation function* $C(i, i+r) = \langle s_i s_{i+r} \rangle$ *with* $i = 1$ *for a long-range Ising chain with* $N = 100$, $\sigma = 5/8$ *and* $\beta J = 1$. *Left: results of four networks with a single* RNN *cell and fully connected layer (see Figure 6) for increasing number of parameters* P_θ. *The* $C(1, 1+r)$ *are estimated with* $n = 10^5$ *samples and compared to an* MCMC *estimate, obtained by a single spin flip algorithm (Appendix A.1). The standard deviations are not shown for the sake of clarity. Right: a network is trained using a single* RNN *cell but a different fully connected layer for every site (see Figure 24) with* $P_\theta = 1335$ *parameters. The* $C(1, 1+r)$ *are depicted with standard deviations.*

5.3 SPIN GLASS

Having studied the impact of site-dependent and long-range interactions, consider the one-dimensional spin glass from section 1.3. The interactions J_{ij} are random variables according to a Gaussian distribution $\mathcal{N}\left(0, C/r_{ij}^{2\sigma}\right)$. This Hamiltonian has site-dependent, long-range and random interactions. The bonds (i, j) are generated as follows [27, 53]: choose an average coordination number z_b. Then generate $Nz_b/2$ (i, j)-couples with probability $p((i, j)) = C/r_{ij}^{2\sigma}$ without replacement, so that every combination (i, j) appears once at the most. On average, every site interacts with z_b other sites. The constant C is calculated as

$$C = \left(\sum_{i<j} \frac{1}{r_{ij}^{2\sigma}} \right)^{-1},$$

so that $\sum_{i<j} p((i, j)) = 1$ is normalised to one. Once the bonds are generated, the interactions are drawn from a Gaussian distribution with mean zero and variance one: $J_{ij} \sim \mathcal{N}(0, 1)$.

Contrary to section 5.2, the total number of interactions is limited to $Nz_b/2$; these interactions are called *dilute interactions* [27]. The energy calculation scales linearly with the system size N so that it is possible to train networks for larger systems without running into memory issues and/or very long run times. This is actually quite important since finite-size effects are very large in these models [27]. Here we only look at smaller system sizes. Disordered systems can

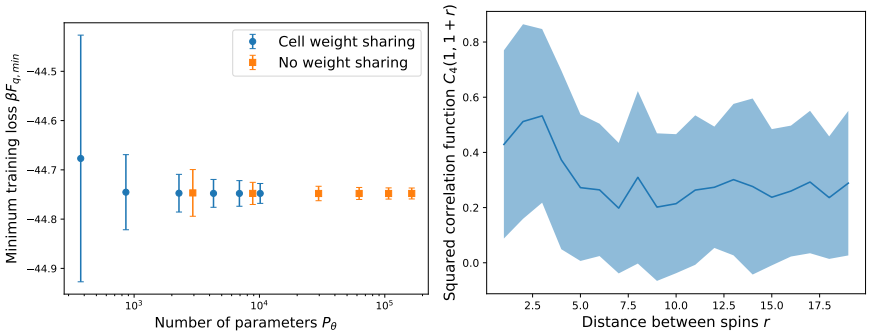

Figure 26: *Results for a one-dimensional spin glass with* $N = 20$, $\sigma = 5/8$, $\beta = 1.35$ *and* $z_b = 6$. *Left: the minimum free energy loss* $\beta F_{q,min}$ *with standard deviation for networks trained with increasing number of parameters* P_θ. *The networks have an* RNN *cell with parameter sharing (blue) or without (orange) and both do not share weights for the fully connected layers. Right: the averaged squared correlations* $C_4(i, i+r) = \left[\langle s_i s_{i+r}\rangle^2\right]$ *with* $i = 1$. *The* $[\ldots]$ *is averaged over ten different* J_{ij} *realisations, while the* $\langle\ldots\rangle$ *is averaged over* $n = 10^6$ *spin configurations from the* RNN.

have multiple pure states in the spin glass phase [53]. If we want the network to capture all of them, we expect temperature annealing to be essential during training.

For disordered systems such as spin glasses, it is common practice to not share weights in an RNN [19]. In [22] there is no weight sharing for both the RNN cell and the fully connected layer Σ. In Figure 26 (left), we compare the minimum loss of an RNN with no weight sharing to the RNN from Figure 24. Both networks find approximately the same minimum training loss when enough parameters are used. This suggests that there is no significant difference between the two.

The nature of short-range spin glasses is not yet fully understood and not many analytical solutions are available. One often uses parallel tempering MC simulations, which are more difficult to implement than the MCMC algorithms considered in this study [32]. Hence, it is more difficult to test whether the network captures the correct physics. We focus on the minimum training loss and the correlations for a short spin chain in the mean-field regime ($\frac{1}{2} < \sigma < \frac{2}{3}$). We train networks with an increasing number of parameters P_θ and compare the minimum training loss, for which the results can be seen in Figure 26. The minimum loss is higher and has a larger standard deviation for a small P_θ than for a larger P_θ. For $P_\theta \gtrsim 10^2$, the minimum loss does not decrease significantly when increasing P_θ. This suggests that the networks find a good minimum of the free energy but this is not necessarily the correct one. We visualise the generated configurations of the RNNs with the least (a high minimum loss) and the largest number of parameters (a low minimum loss) in Figure 27. If the net-

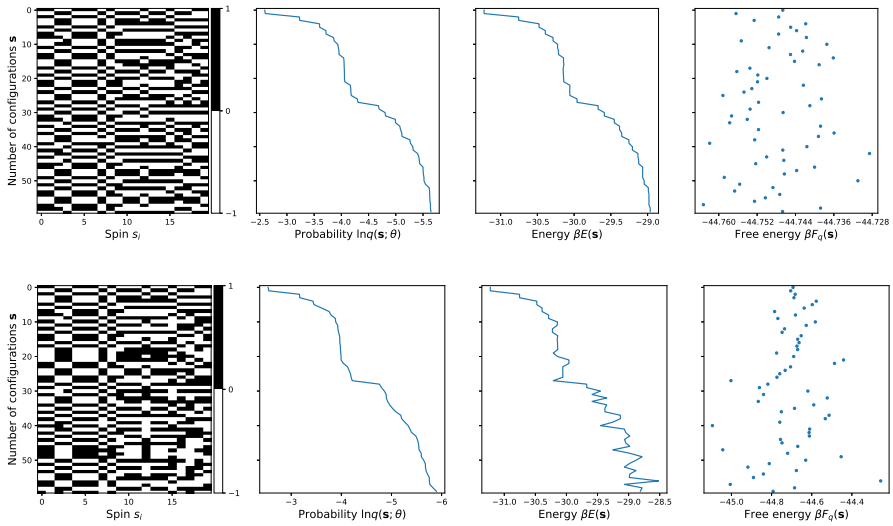

Figure 27: *A visualisation of the generated configurations* $\mathbf{s}$ *for a one-dimensional spin glass with* $N = 20$, $\sigma = 5/8$, $\beta = 1.35$ *and* $z_b = 6$. *The figures represent the different spin chains that were generated by the RNN, sorted according to the network probability* $q(\mathbf{s};\theta)$. *The spins are visualised (left) as black* $(s = +1)$ *and white* $(s = -1)$. *The energies* $\beta E(\mathbf{s})$ *and the free energies* $\beta F(\mathbf{s})$ *are also depicted. The figures represent the* 60 *configurations with the highest network probability. The system has* 2^{20} *different possible configurations. Top: the network is trained with* $P_\theta = 375$ *and has* $\beta F_{q,min} = -44.68 \pm 0.25$. *Bottom: the network has* $P_\theta = 164040$ *and* $\beta F_{q,min} = -44.75 \pm 0.01$.

work distribution approximates the Boltzmann one, then the $q(\mathbf{s};\theta)$ and $\beta E(\mathbf{s})$ curves are similar (see chapter 3) and the free energy has a small variance (zero variance principle). The network with low minimum loss has similar $q(\mathbf{s};\theta)$ and $\beta E(\mathbf{s})$ curves and a smaller variance in the free energy. For the network with a higher minimum loss, the $q(\mathbf{s};\theta)$ and $\beta E(\mathbf{s})$ curves differ more noticeably and the variance of the free energy is larger. These results suggest that the network can capture the distribution of the spin glass system.

The next step is to study the physical properties of the system. We focus on the squared disconnected correlations $C_4(i,j)$, for which an exact solution is available (10). The correlations are averaged over ten different J_{ij} realisations and shown in Figure 26 (right). Unfortunately, there is not much we can deduce from the figure due to the large variances. In practice, the correlations are averaged over many J_{ij} realisations. For example, in [53] the correlations are averaged over 1000 different J_{ij} realisations. However, in the context of this study, this requires too much simulation time and is computationally too expensive for us. If there is access to more computational resources, it is possible to run simulations for longer spin chains and more J_{ij} realisations. As a result, one can perform a more in-depth study of the physical properties of the spin glasses, such as the correlations.

5.4 CONCLUSION

We have studied one-dimensional spin chains, with site-dependent and long-range interactions, by utilising multiple RNN architectures.

An architecture with full parameter sharing does not suffice for the random Ising chain with site-dependent interactions. Under the conditions that the parameters of the fully connected layers are no longer shared, the network can capture site-dependent interactions and the physics of the system.

For power-law interactions, long-range dependencies between spins are not a major concern for the RNN after switching off weight sharing. It still captures the correlations to an accuracy comparable to MCMC estimates without any further adjustments for the long-range dependencies.

The first results for one-dimensional spin glasses are promising. The network finds a low minimum training loss with a small variance, which suggests that the network can learn the Boltzmann distribution. However, a more in-depth study of the physical properties of the system is required to establish RNNs as a valid simulation technique for spin glasses with power-law interactions.

6 CONCLUSION AND OUTLOOK

6.1 CONCLUSION

In this dissertation, we have examined autoregressive Machine Learning (ML) models as an alternative to Monte Carlo (MC) simulations in statistical physics. The models are trained in such a way that the autoregressive ansatz $q(\mathbf{x};\theta)$ approximates the Boltzmann distribution $p(\mathbf{x})$.

In chapter 3, we have trained a network on a set of Boltzmann configurations of the one-dimensional Ising, obtained with Markov Chain Monte Carlo (MCMC). This training approach corresponds to density estimation. A Recurrent Neural Network (RNN), with only a single cell and fully connected layer, successfully captures the physics of the one-dimensional Ising model. The physical and training results do not differ significantly for many combinations of the hyperparameters. Because the physics is the same for every site, the network needs few parameters ($P \lesssim 150$) to achieve results consistent with MCMC. The RNN provides normalised probabilities of the configurations, which can be combined with Neural Importance Sampling (NIS) to obtain unbiased estimations with reliable errors. Especially for smaller data sets, NIS provides better estimates of the physical quantities. While for MCMC the majority of the effort goes into finding efficient sampling algorithms, the RNN efficiently samples independent configurations so that the training of the model is the most expensive part of the simulation.

In chapter 4, we train a network on configurations generated by the network itself. The training is, therefore, more expensive than in chapter 3 but no a priori configurations of the Boltzmann distribution are needed. The RNN architecture can correctly capture the physics of the one-dimensional Ising. In fact, the results are very similar to the ones of chapter 3 with the advantage that the simulation methodology only requires a Hamiltonian. For low temperatures ($T \approx 0$), we perform temperature annealing to prevent mode collapse on the up and down chain. Since the training time scales linearly with the system size and only a relatively small number of parameter updates are needed ($\lesssim 200$), we train spin chains up to $N = 10^4$ on a single Graphics Processing Unit (GPU). In section 4.3, we incorporate the symmetries of the system into the variational ansatz and sampling procedure. For the one-dimensional Ising, imposing symmetry does not add any significant benefit to the training or results.

In chapter 5, we apply the variational method to more complex one-dimensional spin systems with site-dependent and long-range interactions. Since the physics is no longer the same for every site, an RNN with full parameter sharing cannot capture the Boltzmann distribution. Upon using different fully connected layers for every site in the chain, the network can describe the correlations between spins with site-dependent (section 5.1) and power-law (section 5.2) interactions. In particular, with a small number of parameters ($P_\theta \lesssim 1500$ for $N = 100$), the network can find a low minimum loss in relatively few training steps. This is not the case when the network has full parameter sharing. Therefore, it is beneficial to adapt the network architecture to the system under study. The architecture also seems sufficient for one-dimensional spin glasses (section 5.3) but a more in-depth study is recommended.

We conclude that an autoregressive Neural Network (NN) offers a valid alternative to traditional MC simulations for the systems considered here. Whereas the MCMC configurations are weighted by the Boltzmann distribution, an RNN provides normalised probabilities which provide access to quantities that depend on the partition function. Markov Chain Monte Carlo configurations are directly sampled from the Boltzmann distribution – or at least very close to it – and variance estimates are easily obtained by multiple independent runs or by the batch means method. An NN does not always fully capture the distribution, which introduces biases in the estimations. In that case, NIS provides asymptotically unbiased estimates and reliable variances. For many physical systems, there exists a relatively simple MCMC algorithm, for example, the single spin-flip ones for the Ising models. Certain systems, such as spin glasses, do not lend themselves easily to simulations with MCMC techniques. In those cases, ML can offer an alternative simulation methodology. In ML simulations, the majority of the effort goes into designing a suitable architecture and training the model after which configurations can be efficiently sampled without correlations. The complexity of the problem is shifted to solving a high-dimensional and intrinsically hard optimisation process. Therefore, it is beneficial to study the intimate relation between systems and model architectures. If the network is adapted to the system under study, the optimisation is easier and the model requires less training.

6.2 OUTLOOK

Recurrent Neural Networks for simulations in classical statistical physics have only recently been proposed [21]. There is much unknown territory to be explored. In particular, a more thorough study of the relation between the network architecture and the system is the logical next step. In [22] one proposes special constructions to increase

the expressiveness of the model, named *Tensorised RNNs*. They also use recent developments in ML, so-called *Dilated RNNs* [14], to capture long-term dependencies in the system. This could help to simulate systems with very long-range interactions. An understanding of the network-system relation could help to find a more general mechanism for simulating classical many-body systems with ML.

We first discuss some broad possible extensions and ideas. We have focused on one-dimensional spin systems without phase transitions (chapter 3 and chapter 4) or away from the transition at $T < T_c$ (chapter 5). One could extend autoregressive RNNs on a broad range of systems and system parameters, for example:

- Can the model capture phase transitions ($T \approx T_c$) and produce the correct critical exponents?
- How can the one-dimensional input sequence of RNNs be used for higher-dimensional lattice systems? In two dimensions, one can snake through the lattice [21].
- The RNN can be extended to finite-dimensional discrete sample spaces [21]. Can RNNs be combined with infinite-dimensional or continuous state spaces? In that case, the one-hot encoding of the input is no longer straightforward.
- The RNN is per construction open on both sides of the input sequence. Is there a way to implement periodic boundary conditions? Spin glasses, for example, are often studied for periodic or antiperiodic boundary conditions and show different behaviour for open boundary conditions [24].

For this study, in particular, we suggest the following possible improvements and/or extensions:

- The variational approach provides a general scheme to optimise the ML models [55]. A more quantitative convergence measure for the free energy loss is a potential improvement, especially when no analytical solution is available. When does the free energy loss reach the free energy of the system during training? In that way, one knows how well the model approximates the Boltzmann distribution even before using the configurations for estimations. One possibility is to focus on the standard deviation of the free energy loss. A small deviation suggests a well-trained network close to the Boltzmann distribution [55].
- For the one-dimensional Ising model, we have found that the hyperparameters do not significantly impact the results, but this is not necessarily always the case. Aside from the network architecture, a more in-depth study of the effect of the hyperparameters on the training of the model will help to understand why something works or does not.

- In practice, one often performs a finite-size scaling analysis to determine the critical temperature of the system under study. In chapter 5, due to computational and time restrictions, we have chosen the parameters of the simulations based on other research [53]. However, different boundary conditions and finite-size effects could suggest a different critical temperature.

- If there is access to more computational resources, the analysis of the spin glass (section 5.3) can be extended to simulations for longer spin chains and more bond configurations. A more in-depth analysis of the correlations can confirm if the RNN is a viable simulation methodology for the system.

Finally, we have mainly focused on autoregressive RNNs because they are simple and provide normalised probabilities of the configurations. There are many other generative models which can be used for simulations in statistical physics (see Table 2). Similarly, there exist other network architectures, such as a Convolutional Neural Network (CNN), which can also be used for simulations. We do not have to limit ourselves to one specific type of model or network. Instead, one can explore a large number of possibilities in ML and see if there are other more viable options.

Part III

APPENDICES

A

STATISTICAL PHYSICS

A.1 ISING MCMC

This section contains the details of the MCMC implementation for the data generation mentioned in chapter 3. We describe the MCMC algorithm and the details about the data generation.

Consider a one-dimensional Ising chain with N spins and open boundary conditions (7). For this system, a simple but effective single spin-flip MCMC algorithm is chosen [20]:

1. *Initialisation*: choose an initial configuration. The two most common are a completely ordered state ($T \to 0$) with all spins up or down, or a completely random state ($T \to \infty$).

2. *Trial step*: make a trial step by flipping one randomly chosen spin of the system $s_j \to -s_j$, obtaining a new configuration

$$x' = (s_1, s_2, \ldots, -s_j, \ldots, s_N).$$

3. *Acceptance step*: accept the new configuration with a probability that satisfies detailed balance

$$\alpha = \min\left(1, \frac{p(x')}{p(x)}\right) = \min\left(1, \exp\left(-\beta \Delta E(x \to x')\right)\right),$$

with $\Delta E(x \to x')$ the change in the total energy of the system, going from configuration x to x'. Since the system is one-dimensional, the change in energy for a single spin flip is

$$\Delta E(s_j \to -s_j) = 2Js_j(s_{j-1} + s_{j+1}) + 2Hs_j.$$

In order to respect open boundary conditions, one of the two interaction terms is omitted if $j = 1$ or $j = N$. In practice, one generates a random number $r \in [0, 1]$. The new configuration x' is accepted for $r \leqslant \alpha$, while for $r > \alpha$, the new configuration is rejected.

4. *Iteration*: go back to step 2.

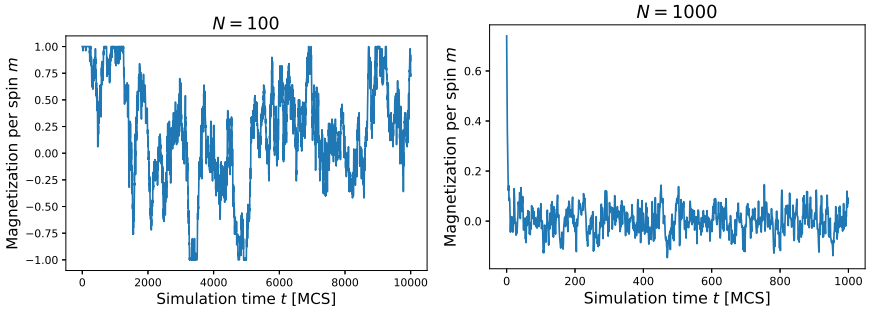

Figure 28: *The running magnetisation for an Ising chain with* $\beta J = 2$ *(left) and* $\beta J = 0.5$ *(right). Both simulations have vanishing magnetic field* $(\beta H = 0)$ *and start from an ordered configuration.*

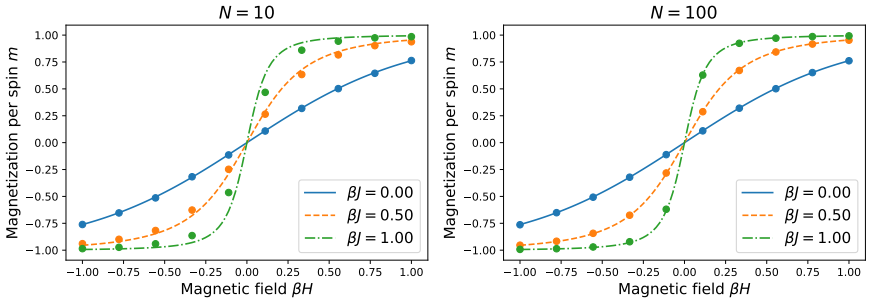

Figure 29: *The magnetisation per spin* m *for the one-dimensional Ising system with* $N = 10$ *(left) and* $N = 100$ *(right). Results of* 100 *independent* MC *simulations, each with* $n = 100$ *samples, are compared with the analytical solution for* $N \to \infty$*. The standard deviations are not visible.*

In practice, not every configuration is used to estimate quantities because subsequent configurations are highly correlated. A frequently used time unit is the Monte Carlo step per spin (MCS), which is equivalent to N spin-flip trials. For the MCMC simulations considered here, time measurements are done in MCS units. A configuration is generated at each MCS which corresponds to N iterations of the MCMC algorithm.

Since MCMC converges to the Boltzmann distribution after an initial number of steps, it is necessary to look at the burn-in period. In Figure 28 the running magnetisation can be found when starting from an ordered configuration $\mathbf{s}_0 = (\uparrow\uparrow \ldots \uparrow)$. In general, the one-dimensional Ising system does not need many steps to reach stationary behaviour, especially for high temperatures. Note that in the absence of a magnetic field $(\beta H = 0)$, the average magnetisation vanishes $(m = 0)$, but the running magnetisation fluctuates around zero.

To check if the MCMC sampling works, the estimated magnetisation is compared to the analytical solution in the thermodynamic limit. This can be seen in Figure 29: the results for different combinations of βJ and βH follow the analytical solution (5). For larger system

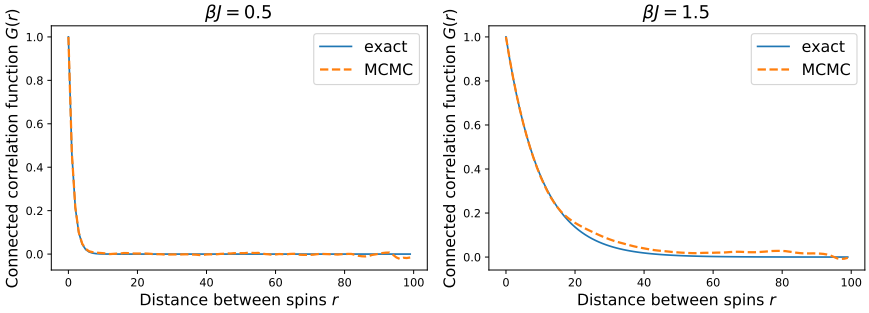

Figure 30: *Comparison of the estimated connected correlation function* $G(r)$*, using* $n = 10^4$ *consecutive samples, with the analytical solution* $(N \to \infty)$*. The one-dimensional Ising has* $N = 100$ *spins and* $\beta H = 0$*.*

sizes, the solution is closer to the analytical solution $N \to \infty$. The analytical solution of the connected correlation function for $\beta H = 0$ is compared to the MCMC estimate in Figure 30. The correlations are averaged over all spin pairs that are a distance r apart. For a system with open boundary conditions, there are many more spin pairs over a small than over a large distance. The correlations are small for a one-dimensional Ising system with weak coupling $(\beta J = 0.5)$. For a stronger coupling $(\beta J = 1.5)$, spins are correlated over longer distances.

It is expected that the performance of the learnt distribution heavily depends on the quality of the data set. In particular, many ML techniques use the i.i.d. assumption: all data points are independently and identically distributed from the real-world distribution. In that case, all spin configurations carry equal weight in the data set. Especially the independence assumption is important for MCMC since this technique inherently provides correlated samples. The cheapest method to get a rough estimate of the correlation time τ is the batch means or data blocking method. If the samples are divided into batches, then the batch means are approximately independent when the batch size m_b is sufficiently large. This is a different batch size than the one used for ML. If we use (12), the error is underestimated because the MCMC samples are correlated. Instead, we should use an estimate of (14). For a large enough batch size, the batch means are approximately independent and the estimate in (12) is appropriate. Figure 31 shows the typical flattening of the error estimate for large batch sizes. The MCMC correlation time is approximated by the position of the elbow. Before the elbow, the batch means are correlated and the variance is underestimated. Correlations between consecutive samples are smaller for a weaker coupling $(\beta J = 0.5)$ than for a stronger coupling $(\beta J = 1.5)$. In practice, we add a configuration to the data set every τ MC steps.

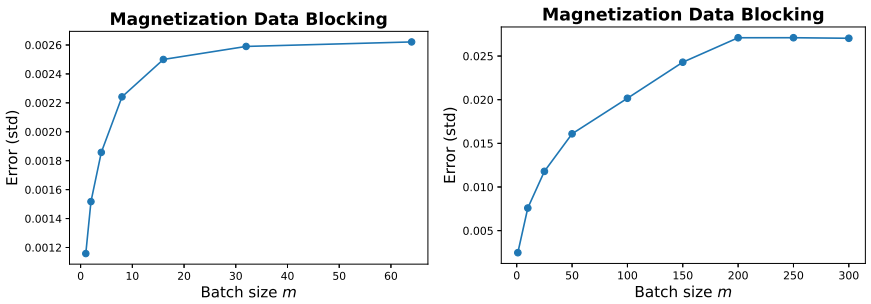

Figure 31: *The batch means method to estimate the correlation time between consecutive samples for MCMC runs. The one-dimensional Ising systems have a weak coupling $\beta J = 0.5$ (left) and stronger coupling $\beta J = 1.5$ (right) and $\beta H = 0$ (both). Samples are approximately uncorrelated when the error estimates start to converge.*

A.2 NEURAL IMPORTANCE SAMPLING

Neural Importance Sampling, introduced by [38], estimates observables O from a physical system with distribution p as

$$\widehat{O}_n = \sum_{i=1}^{n} w_i O(\mathbf{s}_i) \qquad \text{with} \qquad \mathbf{s}_i \sim q(\mathbf{s};\theta),$$

where q is a model distribution which approximates p. The estimation contains n samples from the distribution q, which in this case, is the network distribution. The w_i are the normalised importance weights, obtained by normalising the importance weights $\widetilde{w}_i$:

$$w_i = \frac{\widetilde{w}_i}{\sum_{i=1}^{n} \widetilde{w}_i} \qquad \text{with} \qquad \widetilde{w}_i = \frac{\exp\left(-\beta E(\mathbf{s}_i)\right)}{q(\mathbf{s}_i;\theta)}.$$

Since the normalising factor of the Boltzmann distribution is unknown, this technique is called *self-normalised importance sampling*. Notice the absence of the factor $1/n$ in front of the sum and the presence of the weights w_i, which is not the case for the standard MC estimates. Contrary to MCMC sampling, the samples can be independently and identically sampled from the RNN. For the observables which do not depend on the partition function Z, one can obtain reliable errors as

$$\mathrm{Var}\left(\widehat{O}_n(\mathbf{s})\right) = \frac{\mathrm{Var}\left(O(\mathbf{s})\right)}{n_{\mathit{eff}}},$$

where the variance can be estimated using (24) and n_{eff} is the *effective sample size*. It gives an idea of how many samples, drawn from the distribution $q(\mathbf{s};\theta)$, have the same quality as the samples from $p(\mathbf{s})$. In literature the effective sample size is often estimated as

$\widehat{n}_{\mathit{eff}} = 1/\sum_i w_i^2$ [17]. This expression can be used for the energy of the system, magnetisation per spin and the correlation function from Table 1, since they do not explicitly depend on the partition function. However, for the free energy and the entropy, which both do depend on Z as $O(\mathbf{s}, Z) = g(\mathbf{s}) + h(Z)$, a more complicated version for the errors has to be used [38]:

$$\operatorname{Var}\left(\widehat{O}_n(\mathbf{s}, Z)\right) = \frac{1}{n}\psi^T \left\langle \phi\phi^T \right\rangle_q \psi$$

$$\text{with} \qquad \phi = \begin{pmatrix} g\widetilde{w} - \langle g\rangle_p Z \\ \widetilde{w} - Z \end{pmatrix} \qquad \text{and} \qquad \psi = \begin{pmatrix} 1/Z \\ -\langle g\rangle_p / Z + h'(Z) \end{pmatrix},$$

which can be rewritten as

$$\begin{aligned}\operatorname{Var}\left(\widehat{O}_n(\mathbf{s}, Z)\right) = \frac{1}{n}\Bigg(&\frac{1}{Z^2}\left\langle \left(g\widetilde{w} - \langle g\rangle_p Z\right)^2 \right\rangle_q \\ &+ \left(-\langle g\rangle_p / Z + h'(Z)\right)^2 \left\langle (\widetilde{w} - Z)^2 \right\rangle_q \\ &+ \frac{2}{Z}\left(-\langle g\rangle_p / Z + h'(Z)\right)\left\langle (g\widetilde{w} - \langle g\rangle_p Z)(\widetilde{w} - Z)\right\rangle_q\Bigg).\end{aligned}$$

Here the Boltzmann expectation values $\langle \ldots \rangle_p$ are estimated using (24), the network expectations $\langle \ldots \rangle_q$ using (13) and the partition function Z as

$$\widehat{Z}_n = \frac{1}{n}\sum_{i=1}^{n} \widetilde{w}_i.$$

Direct use of the unnormalised weights $\widetilde{w}_i$ often results in numerical instability and overflows. For this reason, the estimation is rewritten in terms of the normalised importance weights w_i and $\ln \widehat{Z}$. For example, the first term in the equation above is estimated by

$$\begin{aligned}\frac{1}{Z^2}\left\langle \left(g\widetilde{w} - \langle g\rangle_p Z\right)^2 \right\rangle_q &\approx \frac{1}{\widehat{Z}_n^2}\frac{1}{n}\sum_{i=1}^{n}\left(g(\mathbf{s}_i)\widetilde{w}_i - \widehat{g}_n\widehat{Z}_n\right)^2 \\ &\approx \frac{1}{\widehat{Z}_n^2}\frac{1}{n}\sum_{i=1}^{n}\widehat{Z}_n^2\left(g(\mathbf{s}_i)\frac{\widetilde{w}_i}{\widetilde{Z}_n} - \widehat{g}_n\right)^2 \\ &\approx \frac{1}{n}\sum_{i=1}^{n}\left(n g(\mathbf{s}_i) w_i - \widehat{g}_n\right)^2,\end{aligned}$$

where we have used the estimation of the partition function and the definition of the normalised weights (25) in the last line. The estimations for the other terms can be calculated in a similar way, which

leads to the following expression for the estimated variance of the estimations $\widehat{O}_n(s, Z)$:

$$\widehat{\mathrm{Var}}\left(\widehat{O}_n(\mathbf{s}, Z)\right) = \frac{1}{n}\left(\frac{1}{n}\sum_{i=1}^{n}(ng(\mathbf{s}_i)w_i - \widehat{g}_n)^2 + \mathcal{A}^2\frac{1}{n}\sum_{i=1}^{n}(nw_i - 1)^2 + 2\mathcal{A}\frac{1}{n}\sum_{i=1}^{n}(ng(\mathbf{s}_i)w_i - \widehat{g}_n)(nw_i - 1)\right),$$

where $\mathcal{A} = -\widehat{g}_n \pm 1$ for the entropy $(+)$ and the free energy $(-)$. For quantities that do not depend on the partition function $(h = 0)$, the factor becomes $\mathcal{A} = -\widehat{g}_n$.

B

LEARNING WITH DATA

B.1 NETWORK ARCHITECTURE

This section contains a more in-depth description of the network architecture used in chapter 3 and chapter 4; and a derivation of the probability output (20).

An important thing to consider first is the input of the network. Neural Networks are very powerful and able to learn things that are not meant to be learnt. For example, if the values of the spins ($s = +1$ or $s = -1$) are directly used as input, the network might learn something about the ordering of the spin values. Since this is generally not desirable, it is a common practice in ML to treat each input value equally. For discrete value spaces such as the Ising model, this is done by *one-hot encoding* the inputs of the network: here the value $s = -1$ is encoded to the vector $\mathbf{s} = (1, 0)$ and $s = +1$ to $\mathbf{s} = (0, 1)$.

To learn the first probability $p(s_1)$, the network has to be initialised by an *artificial* one-hot encoded spin $\mathbf{s}_0$ and a hidden vector $\mathbf{h}_0$, both chosen as zero vectors [21]. The inputs $\mathbf{s}_0$ and $\mathbf{h}_0$ are sent through an RNN cell, which operates on the inputs. The simplest RNN cell is a weighted sum sent through a non-linear activation function: this is just the perceptron from (18). The output is treated as a new hidden vector

$$\mathbf{h}_1 = \sigma\left(W\left[\mathbf{h}_0; \mathbf{s}_0\right] + \mathbf{b}\right).$$

The hidden vector $\mathbf{h}_1$ is both used to predict the probability $p(s_1)$ and to store the information of the chain, which is then used as input together with s_2 to predict $p(s_2 \mid s_1)$. The hidden vectors can be interpreted as probabilities by first sending them through a fully connected layer Σ to get the desired dimensions for the output. Then it is passed through a *softmax activation function* σ. This normalises and squishes the output values in the range $[0, 1]$, which can then be interpreted as a probability distribution $\mathbf{y}_1$. The output vectors have the same dimension as the input vectors: a probability for a spin down and a probability for a spin up. The conditional probability is then calculated as a dot-product between inputs and outputs:

$$q_\theta(s_1) = \mathbf{s}_1 \cdot \mathbf{y}_1 = \mathbf{s}_1 \cdot \begin{pmatrix} q_\theta\left(\mathbf{s}_1 = (1, 0)\right) \\ q_\theta\left(\mathbf{s}_1 = (0, 1)\right) \end{pmatrix}.$$

This process is repeated until the hidden vector $\mathbf{h}_{N-1}$ and spin $\mathbf{s}_{N-1}$ are used to predict $p(s_N \mid s_{N-1}, \ldots, s_1)$.

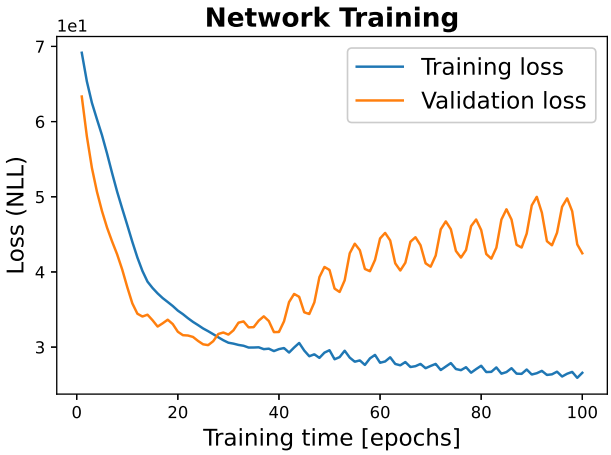

Figure 32: *Training and validation loss (NLL) for a network trained on* $n = 2$ *spin chains* ($N = 100$). *The network has* $n_l = 1$ *hidden layer and a hidden dimension of* $h_d = 25$ *with a total of* $P_\theta = 2227$ *parameters. The network is learning since the training loss decreases: this is a good initial sign of a working algorithm.*

$$q(\mathbf{s};\theta) = \prod_{i=1}^{N} q_\theta(s_i \mid s_{i-1},\dots,s_1) = \prod_{i=1}^{N} \mathbf{s}_i \cdot \mathbf{y}_i.$$

B.2 TRAINING THE NETWORK

In this section, the training approach from chapter 3 is elaborately discussed: this includes testing the network for expected performance and other details.

Before performing a full-scale analysis on the Ising model, it is always a good idea to check if the network works. One simple method is to train the network on a very small data set. If the architecture and the training steps are correctly implemented, any network can minimise the loss of very few samples, given that the model is powerful enough [19]. The numerical implementation of the program is inspired by [12] in combination with *PyTorch* [39] as DL library. The network is then trained using a small data set of $n = 4$ samples, two for training and two for validating, for a system with $\beta J = 1$, $\beta H = 0$ and $N = 100$. Figure 32 shows that the loss decreases after multiple epochs of training. This is an initial sign that the network works and learns something. One can also notice that the validation loss starts to increase after several epochs while the training loss keeps decreasing. This implies that the network is starting to overfit to the data set of two samples.

The NLL loss does not directly provide quantitative information about how well the network is learning the Boltzmann distribution. Therefore the Boltzmann weights $\exp(-\beta E(\mathbf{s}_i))$ are used as a measure

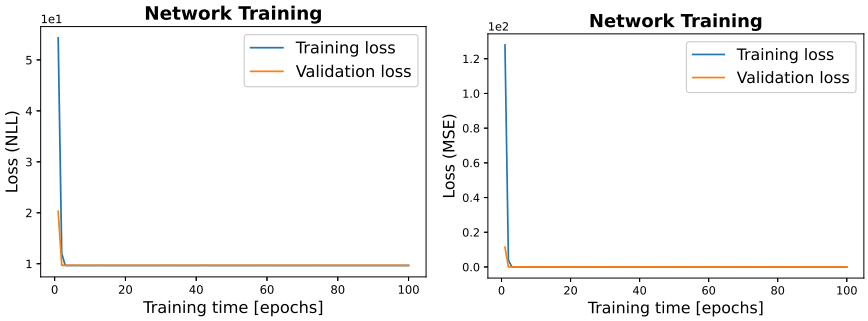

Figure 33: *Training and validation* NLL *loss (left) and* MSE *metric (right) for a network trained on* $n = 10^4$ *generated spin chains* ($N = 100$). *The network has* $n_l = 1$, $h_d = 2$ *with a total of* $P_\theta = 31$ *parameters. Both the* NLL *and the* MSE *are minimised during training, which implies that they both measure the similarity between the learnt distribution and the Boltzmann one. However, the minimum* NLL *depends on the system and the amount of data, while the* MSE *approaches zero.*

for learning the correct distribution. While the original data only has access to the unnormalised Boltzmann weights, the network provides normalised probabilities $q(\mathbf{s}_i;\theta)$. We consider the MSE of the logarithms of the ratios as a performance metric, which is a simple solution that prevents extremely large numbers. For an even number of batch samples n_b, it is calculated as

$$\mathrm{MSE}(\theta) = \frac{2}{n_b}\sum_{i=0}^{\frac{n_b}{2}-1}\left(\ln\left(\frac{q(\mathbf{s}_{2i};\theta)}{q(\mathbf{s}_{2i+1};\theta)}\right) - \ln\left(\frac{\exp(-\beta E(\mathbf{s}_{2i}))}{\exp(-\beta E(\mathbf{s}_{2i+1}))}\right)\right)^2 .$$

In Figure 33 the NLL loss and the performance metric are compared to each other. While the minimum NLL is around ≈ 10, a value from which it is difficult to derive if the network has learnt the distribution, the minimum MSE metric is $\lesssim 10^{-2}$. If the network can reproduce the Boltzmann distribution completely, the performance metric is zero: a small MSE implies that the network distribution is close to the Boltzmann distribution.

C

VARIATIONAL LEARNING

C.1 TRAINING THE NETWORK

This section elaborately describes the training approach from chapter 4: this includes the practical numerical implementation and the details of the training approach.

When the network is trained without an explicitly available data set, configurations are generated by the network and the variational free energy is minimised as a loss function:

$$\beta F_q = \langle \beta E(\mathbf{s}) - \ln q(\mathbf{s};\theta) \rangle_{\mathbf{s}\sim q} \,,$$

with $E(\mathbf{s})$ the energy of the one-dimensional Ising chain. In practice, the expectation value is approximated by an MC estimate

$$L(\theta) = \frac{1}{n_b} \sum_{i=1}^{n_b} \left(\beta E(\mathbf{s}_i) - \ln q(\mathbf{s}_i;\theta) \right).$$

Since the training loss (28) is an estimation of the true loss (27), one has to be careful when back-propagating the loss function. Back-propagation should use the gradients of the true loss, which can be calculated as an expectation value according to the variational distribution

$$\nabla_\theta \beta F_q = \langle \nabla_\theta \ln q(\mathbf{s};\theta) \left(\beta E(\mathbf{s}) + \ln q(\mathbf{s};\theta) \right) \rangle_{\mathbf{s}\sim q} \,. \tag{32}$$

This derivation makes use of the fact that

$$\langle \nabla_\theta \ln q(\mathbf{s};\theta) \rangle_{\mathbf{s}\sim q} = \sum_{\mathbf{s}} q(\mathbf{s};\theta) \nabla_\theta \ln q(\mathbf{s};\theta) = \nabla_\theta \sum_{\mathbf{s}} q(\mathbf{s};\theta) = 0,$$

since the network distribution is normalised per construction and the sum over the configurations can be pulled through the gradients. The gradients in (32) can also be approximated by a standard MC estimate

$$\widehat{\nabla_\theta \beta F_q} = \frac{1}{n_b} \sum_{i=1}^{n_b} \nabla_\theta \ln q(\mathbf{s}_i;\theta) \left(\beta E(\mathbf{s}_i) + \ln q(\mathbf{s}_i;\theta) \right). \tag{33}$$

Notice that the gradients of the estimated loss from (28) are not the same as (33). This is the reason one has to be careful when implementing the loss.

In practice, (28) is measured during training while (33) is used to back-propagate through the network and update the parameters. For PyTorch in particular, this is implemented by setting the argument `(requires_grad=True)` for the $\ln q(\mathbf{s}_i;\theta)$ tensor before the brackets and setting `(requires_grad=False)` for the two terms between the brackets in (33). As a result, a call to `loss.backwards()` only back-propagates through the first $\ln q(\mathbf{s}_i;\theta)$ whose derivative has to be taken. An alternative but equivalent option is to rewrite the expression into a total derivative and perform back-propagation on the expression inside the total derivative.

Since the training loss βF_q is an estimate for the exact free energy of the system, it is quite simple to check if the network training performs as expected by comparing the running training loss with the exact value for $N \to \infty$, which is known for the one-dimensional Ising system. Due to finite-size effects, the analytical solution is not the same as the estimated loss but it already gives an idea of how close the distribution is. When no exact solution for the free energy is available, it is still possible to get an idea of how closely the network distribution $q(\mathbf{s};\theta)$ approximates the Boltzmann distribution $p(\mathbf{s})$. The idea behind this is based on the variational free energy in (27) as an estimation over the distribution $\beta E(\mathbf{s}) + \ln q(\mathbf{s};\theta)$. If the network distribution exactly describes the Boltzmann distribution $q(\mathbf{s};\theta) = p(\mathbf{s})$, then the estimated free energy has zero variance since

$$\beta E(\mathbf{s}) + \ln q(\mathbf{s};\theta) = \beta E(\mathbf{s}) + \ln p(\mathbf{s}) = -\ln(Z),$$

where the partition function Z is just a normalisation constant. The second equality comes from the definition of the Boltzmann distribution (1). Conversely, if the estimator has zero variance, then the distribution is constant: this is called the *zero variance principle*. Unfortunately, this constant does not necessarily need to be the partition function Z. However, if we can prevent mode collapse, then the constant is equal to $-\ln(Z)$ and the variational distribution is the Boltzmann distribution. Consequently, a small variance implies that the variational distribution is really close to the Boltzmann distribution $q(\mathbf{s};\theta) \approx p(\mathbf{s})$ [55].

In Figure 34 (left) a network is trained for a system at high temperature ($\beta J = 0.3$), to avoid mode collapse, using this training approach. As can be seen, learning is unstable since the estimated loss heavily fluctuates. Instead, the network jumps around in the parameter space looking for a minimum variational free energy. The relatively large standard deviation suggests that the variational distribution is not close to the Boltzmann distribution. This behaviour is caused by the

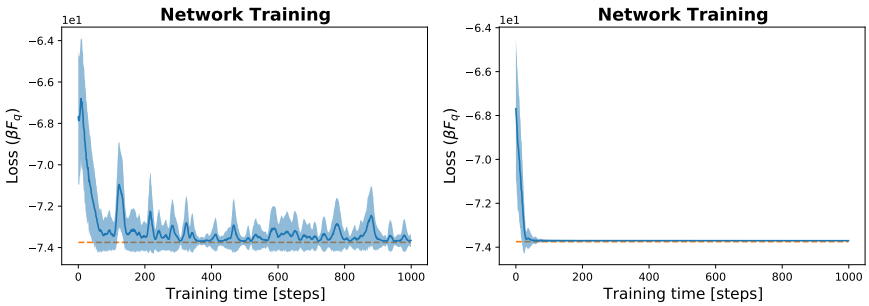

Figure 34: *The network training for a system with* $N = 100$, $\beta J = 0.3$ *and* $\beta H = 0$. *The network has* $P_\theta = 147$ *parameters and is trained without a variance reducing term (left) and with a variance reducing term (right). Each training step corresponds to one parameter update of the network or equivalently, one iteration from Figure 16. Learning is much more stable when the variance reducing is included in the loss function.*

difference in magnitudes of the two terms between brackets in (28), namely $\beta E(\mathbf{s})$ and $\ln q(\mathbf{s};\theta)$ [55].

This problem can be solved by introducing a variance reducing term in the gradients, which has no impact on the true loss function (27) [33]. The variance reducing term fixes the difference in magnitude between the two terms in the gradients. The estimates for the gradients then become [55]

$$\widehat{\nabla_\theta \beta F_q} = \frac{1}{n_b}\sum_{i=1}^{n_b} \nabla_\theta \ln q(\mathbf{s}_i;\theta) \left(\beta E(\mathbf{s}_i) + \ln q(\mathbf{s}_i;\theta) - \frac{1}{n_b}\sum_{j=1}^{n_b}\left(\beta E(\mathbf{s}_j) + \ln q(\mathbf{s}_j;\theta)\right)\right).$$

Figure 34 (right) shows the same network training as before but including this variance reduction. Learning is much more stable: the estimated loss fluctuates much less around the minimum. The estimated variance of the variational distribution is also much smaller, which suggests that the final distribution is much closer to the Boltzmann distribution than without variance reduction.

As mentioned before, the principle of zero variance and the training approach only works if the network does not collapse on certain modes during training. For example, in Figure 35 a network is trained at low temperature ($\beta J = 3$). The magnetisation spectrum should be dominated by two different modes: a spin up and a spin down chain. However, when looking at the generated configurations of the network, only one mode is present (here the spin up chain). Despite a small variance of the variational distribution, the network is unable to describe the correct physics due to mode collapse. It is possible to solve this problem by carefully annealing the temperature from a

high temperature β_0^* to the temperature of the system β [22, 55]. At $\beta_0^* = 0$, the free energy loss is dominated by the entropy term and the system is in a random state. By gradually lowering the temperature, gradient descent explores the different modes that minimise the free energy loss. Intuitively, temperature annealing is a physical way of balancing exploitation and exploration in the parameter space. If training starts at $\beta \to \infty$, the network only finds one mode (spin up chain) and no longer searches for other ones. There is only exploitation and no exploration. Two different annealing schemes are tested:

- *Linear scheme* [22]: $\beta^*(t) = \beta^*(t = 0) + t\Delta t\beta = t\Delta t\beta$, where $\beta^*(t = 0) = \beta_0^* = 0$ is the initial annealing temperature and Δt a measure for the speed of annealing. After $t = 1/\Delta t$ annealing steps, the training has reached the system temperature. Annealing should be slow enough to prevent mode collapse but fast enough so that training does not become computationally too expensive.

- *Power scheme* [55]: $\beta^*(t) = \beta(1-\beta_A^t)$ with $0 < \beta_A < 1$ a measure for the speed of convergence to the system temperature β. If $\beta_A \lesssim 1$, the system needs many annealing steps to converge to the system temperature. The system only reaches the system temperature β at $t \to \infty$.

These two annealing schemes are compared to each other, for which the results can be seen in Figure 35. Contrary to a naive network without annealing, both schemes can prevent mode collapse of the network distribution and estimate the correct correlations in the chain at low temperatures.

Numerical simulations show that both schemes break down if too few or no annealing steps are taken. For example, consider an Ising chain with $N = 100$, $\beta J = 2$ and $\beta H = 0$. Then the power scheme breaks down and has mode collapse for $\beta_A \lesssim 0.8$ when training with $N_T = 10$ parameter updates per temperature. On the other hand, the linear scheme breaks down at $1/\Delta t = N_A \approx 10$ annealing steps when training with $N_T = 10$ parameter updates per temperature. These breakdown regimes depend on the system and training parameters. In particular, when the network is better trained at each temperature (larger N_T), there are fewer annealing steps necessary to prevent mode collapse (smaller N_A).

Since the primary goal of temperature annealing is preventing mode collapse, it does not really matter which of the two schemes is used: both work as long as enough annealing steps N_A are taken for the number of training steps per temperature N_T.

The computational least expensive and simplest option is in this case the linear scheme since there is no need to perform enough annealing steps to converge to the temperature of the system. Instead, the number of annealing steps immediately anneals the temperature

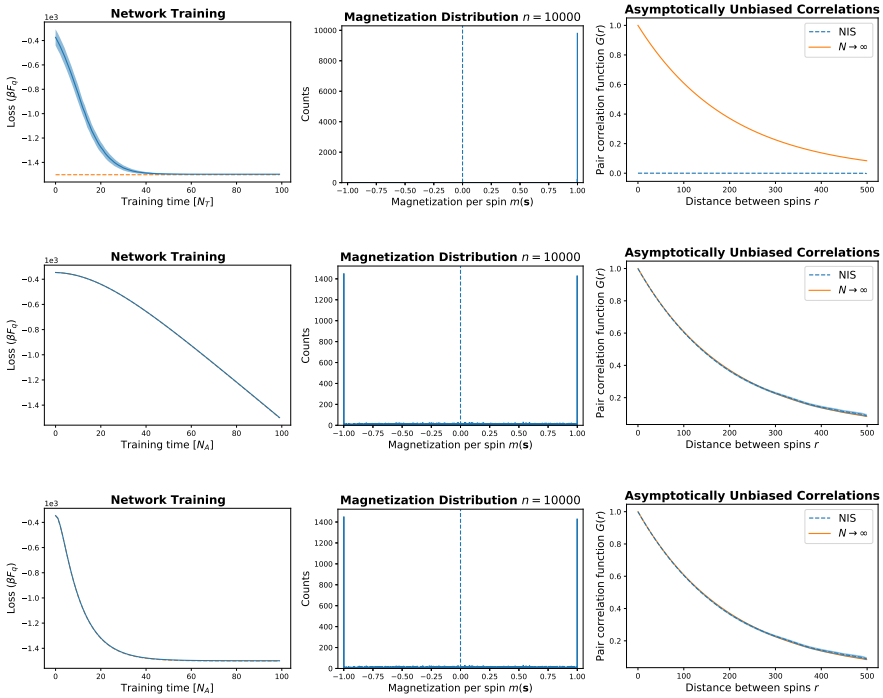

Figure 35: *A comparison between networks* ($P = 147$) *trained with no temperature annealing (top row), annealing with a linear scheme (middle row) and with a power scheme (bottom row). The network training without annealing (top left) is trained for* $N_T = 100$ *parameter updates. The networks with annealing are trained at* $N_A = 100$ *different temperatures from* $\beta_0^* J = 0 \to \beta J = 3$*, each for* $N_T = 100$ *parameter updates and with batch size* $n_b = 1000$*. Every step in the figures is the average training loss of* $N_T = 100$ *parameter updates. Contrary to the network without annealing, both annealing schemes prevent mode collapse. The magnetisation histograms (middle) and correlations (right) are calculated using* $n = 10^4$ *samples from the RNN. The one-dimensional Ising has* $N = 500$*,* $\beta J = 3$ *and* $\beta H = 0$*.*

in N_A steps to the desired temperature, which for this system, in particular, can be very small $N_A \lesssim 10$. However, when few training steps are performed at each annealed temperature, it is beneficial to train the network further at the final temperature of the system. Generally, the training of the networks goes as follows: anneal the temperature linearly for N_A steps, at each step the network is trained for a relatively small number of parameter updates N_T, and then train the network extra at the final β for N_F steps. An example can be seen in Figure 17.

C.2 IMPOSING SYMMETRY

Suppose $q(\mathbf{s};\theta)$ is the variational ansatz for the Boltzmann distribution $p(\mathbf{s})$, which is symmetrised according to the $\mathbb{Z}_2$ symmetry of the system so that [29, 55]

$$q_{\text{sym}}(\mathbf{s};\theta) = \frac{1}{2}\left(q(\mathbf{s};\theta) + q(-\mathbf{s};\theta)\right).$$

The dissimilarity between the variational and Boltzmann distribution is given by (26), which is minimised by an NN to find the best possible parameters θ. In that case, we know that

$$\begin{aligned}
&D_{KL}(q_{\text{sym}} \,\|\, p) \\
&= \sum_{\mathbf{s}} q_{\text{sym}}(\mathbf{s};\theta) \ln\left(\frac{q_{\text{sym}}(\mathbf{s};\theta)}{p(\mathbf{s})}\right) \\
&= \frac{1}{2}\left(\sum_{\mathbf{s}} q(\mathbf{s};\theta) \ln\left(\frac{q_{\text{sym}}(\mathbf{s};\theta)}{p(\mathbf{s})}\right) + \sum_{\mathbf{s}} q(-\mathbf{s};\theta) \ln\left(\frac{q_{\text{sym}}(\mathbf{s};\theta)}{p(\mathbf{s})}\right)\right) \\
&= \frac{1}{2}\left(\sum_{\mathbf{s}} q(\mathbf{s};\theta) \ln\left(\frac{q_{\text{sym}}(\mathbf{s};\theta)}{p(\mathbf{s})}\right) + \sum_{\mathbf{s}} q(-\mathbf{s};\theta) \ln\left(\frac{q_{\text{sym}}(-\mathbf{s};\theta)}{p(-\mathbf{s})}\right)\right) \\
&= \sum_{\mathbf{s}} q(\mathbf{s};\theta) \ln\left(\frac{q_{\text{sym}}(\mathbf{s};\theta)}{p(\mathbf{s})}\right),
\end{aligned}$$

where we have used the symmetry of q_{sym} and p under spin flips. In the last step, the two sums are identical since they are both an enumeration over all possible configuration. The same can be proven if there are additional symmetries such as the reflection symmetry. This equality can be used to prove that

$$
\begin{aligned}
&D_{KL}(q \parallel p) - D_{KL}(q_{sym} \parallel p) \\
&= \sum_{\mathbf{s}} q(\mathbf{s};\theta) \ln\left(\frac{q(\mathbf{s};\theta)}{p(\mathbf{s})}\right) - \sum_{\mathbf{s}} q_{sym}(\mathbf{s};\theta) \ln\left(\frac{q_{sym}(\mathbf{s};\theta)}{p(\mathbf{s})}\right) \\
&= \sum_{\mathbf{s}} q(\mathbf{s};\theta) \left(\ln q(\mathbf{s};\theta) - \ln p(\mathbf{s}) - \ln q_{sym}(\mathbf{s};\theta) + \ln p(\mathbf{s})\right) \\
&= \sum_{\mathbf{s}} q(\mathbf{s};\theta) \ln\left(\frac{q(\mathbf{s};\theta)}{q_{sym}(\mathbf{s};\theta)}\right) \\
&= D_{KL}(q \parallel q_{sym}) \geqslant 0,
\end{aligned}
$$

where the definition of the KL divergence is used and the inequality holds because of the Jensen inequality. Using once again (26) to write $D_{KL}(q \parallel p) - D_{KL}(q_{sym} \parallel p) = \beta F_q - \beta F_{q_{sym}}$, above inequality turns into $\beta F_q \geqslant \beta F_{q_{sym}}$.

This suggests that the variational free energy of the symmetrised model is a lower upper bound for the true free energy βF of the system. Since the variational free energy has to be minimised to approximate the Boltzmann distribution, exploiting the symmetry could result in a better description of the system [29].

D

SIMULATION DETAILS

	SYSTEM PARAMETERS			SIMULATION PARAMETERS					
	N	βJ	βH	method	n	N_{eq}	N_{mc}	τ	init
Fig. 9; Fig. 10; Fig. 11	10	0.8	0	exact	5000	✗	✗	✗	✗
	50	0.8	0	MCMC	5000	100	$2 \cdot 10^5$	40	random
Fig. 12	$10 - 1000$	0.5	0	MCMC	10000	1000	10^4	1	random
Fig. 13; Fig. 14; Tab. 5	100	2	0	MCMC	10000	1000	$4 \cdot 10^6$	400	random
Fig. 15	100	0.8	0	MCMC	10000	100	$4 \cdot 10^5$	40	random

Table 8: *A summary of the system and simulation parameters that were used to simulate the data sets presented in chapter 3. The* exact *method refers to simulations where the configurations are drawn with the exact Boltzmann probabilities, obtained by enumerating over all possible configurations of the one-dimensional Ising.* Init *refers to the initial configuration for the Markov chain. The symbol n stands for the number of samples in the data set. The other symbols can be found in the Table of Symbols.*

	SYSTEM PARAMETERS				NETWORK & TRAINING HYPERPARAMETERS						
	N	βJ	βH	n	h_d	n_l	cell	n_b	η	c_g	epochs
Fig. 9	$10, 50$	0.8	0	5000	$25 - 75$	$1 - 3$	GRU	500	0.01	1000	100
Fig. 10	$10, 50$	0.8	0	5000	25	1	GRU	$10 - 1000$	0.01	1000	100
Fig. 11	$10, 50$	0.8	0	5000	25	1	GRU	500	$0.0001 - 0.1$	1000	100
Fig. 12	$10 - 1000$	0.5	0	10^4	1	1	GRU	500	0.01	2	100
Fig. 13; Fig. 14; Tab. 5	100	2	0	10^4	1	2	GRU	50	0.01	2	100
Fig. 15	50	0.8	0	10^4	$5 - 25$	1	GRU	$10 - 1000$	$0.1 - 0.001$	2	100

Table 9: *A summary of the system parameters and the network and training hyperparameters that were used to obtain the results presented in chapter 3. The symbol n stands for the number of samples in the data set. The other symbols can be found in the Table of Symbols.*

	SYSTEM PARAMETERS			NETWORK & TRAINING HYPERPARAMETERS								
	N	βJ	βH	h_d	n_l	cell	n_b	η	c_g	N_A	N_F	N_T
Fig. 17	500	3	0	5	1	GRU	1000	0.01	2	100	10	10
Fig. 18 (left)	$10^2 - 10^4$	2	0	5	1	GRU	1000	0.01	2	10	10	10
Fig. 18 (right)	100	0.5	0	$1 - 200$	1	GRU	1000	0.01	2	10	10	10
Tab. 6	100	2	0	5	1	GRU	1000	0.01	2	20	10	100
Fig. 19; Fig. 20	10^4	4	0	5	1	GRU	1000	0.01	2	10	10	10
Fig. 21	10	4	0	5	1	GRU	1000	0.01	2	100	20	100
Tab. 7	10	1	0	5	1	GRU	1000	0.01	2	10	10	10

Table 10: *A summary of the system parameters and the network and training hyperparameters that were used to obtain the results presented in chapter 4. The other symbols can be found in the Table of Symbols.*

	SYSTEM PARAMETERS				NETWORK & TRAINING HYPERPARAMETERS									
	N	β	σ	z_b	h_d	n_l	cell	share cell	share Σ	n_b	η	c_g	N_A	N_F	N_T
Fig. 23 (left)	100	10	✗	✗	$5-100$	1	GRU	True	True	10^4	0.01	2	100	100	10
Fig. 23 (right)	100	10	✗	✗	5	1	GRU	True	False	10^4	0.01	2	100	100	10
Fig. 25 (left)	100	1	5/8	✗	$5-100$	1	GRU	True	True	10^4	0.01	2	100	100	10
Fig. 25 (right)	100	1	5/8	✗	5	1	GRU	True	False	10^4	0.01	2	100	100	10
Fig. 26 (left)	20	1.35	5/8	6	$5-50$	1	GRU	True-False	False	10^4	0.01	2	100	100	10
Fig. 26 (right)	20	1.35	5/8	6	5	1	GRU	False	False	10^4	0.01	2	100	100	10
Fig. 27 (top)	20	1.35	5/8	6	50	1	GRU	False	False	10^4	0.01	2	100	100	10
Fig. 27 (bot)	20	1.35	5/8	6	5	1	GRU	True	False	10^4	0.01	2	100	100	10

Table 11: *A summary of the system parameters and the network and training hyperparameters that were used to obtain the results presented in chapter 5. The columns* share cell *and* share Σ *indicate if the* RNN *shares the parameters for the cell and/or fully connected layer* Σ*. The other symbols can be found in the Table of Symbols.*

BIBLIOGRAPHY

[1] M. S. Albergo, Kanwar and P. E. Shanahan. 'Flow-based generative models for Markov chain Monte Carlo in lattice field theory'. In: *arXiv* (2019), pp. 1–13. ISSN: 23318422. DOI: 10.1103/physrevd.100.034515.

[2] Ethem Alpaydin. *Introduction to Machine Learning*. MIT Press, 2014, pp. 1–613. ISBN: 978-0-262-02818-9.

[3] R. A. Baños, L. A. Fernandez, V. Martin-Mayor and A. P. Young. 'Correspondence between long-range and short-range spin glasses'. In: *Physical Review B - Condensed Matter and Materials Physics* 86.13 (2012), pp. 1–14. ISSN: 10980121. DOI: 10.1103/PhysRevB.86.134416.

[4] Joseph W. Britton, Brian C. Sawyer, Adam C. Keith, C. C.Joseph Wang, James K. Freericks, Hermann Uys, Michael J. Biercuk and John J. Bollinger. 'Engineered two-dimensional Ising interactions in a trapped-ion quantum simulator with hundreds of spins'. In: *Nature* 484.7395 (2012), pp. 489–492. ISSN: 00280836. DOI: 10.1038/nature10981.

[5] Peter Broecker, Juan Carrasquilla, Roger G. Melko and Simon Trebst. 'Machine learning quantum phases of matter beyond the fermion sign problem'. In: *Scientific Reports* 7.1 (2017), pp. 1–10. ISSN: 20452322. DOI: 10.1038/s41598-017-09098-0.

[6] Sergio A. Cannas and Francisco A. Tamarit. 'Long-range interactions and nonextensivity in ferromagnetic spin models'. In: *Physical Review B - Condensed Matter and Materials Physics* 54.18 (1996), pp. 1–4. ISSN: 1550235X. DOI: 10.1103/PhysRevB.54.R12661.

[7] John L. Cardy. *Finite-Size Scaling*. Vol. 2. Elsevier, 1988, pp. 1–465. ISBN: 0 444 87109 8.

[8] Giuseppe Carleo. 'The Variational Playground for Machine Learning in Statistical Physics'. In: *Journal Club for Condensed Matter Physics* (Dec. 2018), pp. 1–4. URL: https://www.condmatjclub.org/?p=3498.

[9] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto and Lenka Zdeborová. 'Machine learning and the physical sciences'. In: *Reviews of Modern Physics* 91.4 (2019), pp. 1–13. ISSN: 15390756. DOI: 10.1103/RevModPhys.91.045002.

[10] Giuseppe Carleo and Matthias Troyer. 'Solving the quantum many-body problem with artificial neural networks'. In: *Science* 355.6325 (2017), pp. 602–606. ISSN: 10959203. DOI: 10.1126/science.aag2302.

[11] Juan Carrasquilla and Roger G. Melko. 'Machine learning phases of matter'. In: *Nature Physics* 13.5 (2017), pp. 431–434. ISSN: 17452481. DOI: 10.1038/nphys4035.

[12] Juan Carrasquilla and Giacomo Torlai. 'Neural networks in quantum many-body physics: a hands-on tutorial'. In: *arXiv* (2021), pp. 1–21. URL: http://arxiv.org/abs/2101.11099.

[13] Juan Carrasquilla, Giacomo Torlai, Roger G. Melko and Leandro Aolita. 'Reconstructing quantum states with generative models'. In: *arXiv* (2018), pp. 1–29. ISSN: 23318422. DOI: 10.1038/s42256-019-0028-1.

[14] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark Hasegawa-Johnson and Thomas S. Huang. 'Dilated recurrent neural networks'. In: *Advances in Neural Information Processing Systems* 2017-Decem.Nips (2017), pp. 77–87. ISSN: 10495258. URL: https://arxiv.org/abs/1710.02224.

[15] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho and Yoshua Bengio. 'Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling'. In: *arXiv* (2014), pp. 1–9. URL: http://arxiv.org/abs/1412.3555.

[16] Freeman J. Dyson. 'Existence of a phase-transition in a one-dimensional Ising ferromagnet'. In: *Communications in Mathematical Physics* 12.2 (1969), pp. 91–107. ISSN: 00103616. DOI: 10.1007/BF01645907.

[17] Víctor Elvira, Luca Martino and Christian P. Robert. 'Rethinking the effective sample size'. In: *arXiv* (2018), pp. 1–16. ISSN: 23318422. URL: https://arxiv.org/abs/1809.04129.

[18] Emilio Flores-Sola, Martin Weigel, Ralph Kenna and Bertrand Berche. 'Cluster Monte Carlo and dynamical scaling for long-range interactions'. In: *European Physical Journal: Special Topics* 226.4 (2017), pp. 581–594. ISSN: 19516401. DOI: 10.1140/epjst/e2016-60338-3.

[19] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016. ISBN: 9780262035613. URL: http://www.deeplearningbook.org.

[20] Harvey Gould and Jan Tobochnik. *Statistical and Thermal Physics: With Computer Applications*. Princeton University Press, 2010, pp. 1–552.

[21] Mohamed Hibat-Allah, Martin Ganahl, Lauren E. Hayward, Roger G. Melko and Juan Carrasquilla. 'Recurrent neural network wavefunctions'. In: *arXiv* (2020), pp. 1–19. ISSN: 23318422. DOI: 10.1103/physrevresearch.2.023358.

[22] Mohamed Hibat-Allah, Estelle M. Inack, Roeland Wiersema, Roger G. Melko and Juan Carrasquilla. 'Variational Neural Annealing'. In: *arXiv* (2021), pp. 1–19. URL: http://arxiv.org/abs/2101.10154.

[23] Helmut G. Katzgraber, Matteo Palassini and A. P. Young. 'Monte Carlo simulations of spin glasses at low temperatures'. In: *Physical Review B - Condensed Matter and Materials Physics* 63.18 (2001), pp. 1–10. ISSN: 1550235X. DOI: 10.1103/PhysRevB.63.184422.

[24] Helmut G. Katzgraber and A. P. Young. 'Monte Carlo simulations of spin glasses at low temperatures: Effects of free boundary conditions'. In: *Physical Review B - Condensed Matter and Materials Physics* 65.21 (2002), pp. 1–6. ISSN: 01631829. DOI: 10.1103/PhysRevB.65.214402.

[25] Diederik P. Kingma and Jimmy Lei Ba. 'Adam: A method for stochastic optimization'. In: *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings* (2015), pp. 1–15. URL: https://arxiv.org/abs/1412.6980.

[26] Yann Lecun, Yoshua Bengio and Geoffrey Hinton. 'Deep learning'. In: *Nature* 521.7553 (2015), pp. 436–444. ISSN: 14764687. DOI: 10.1038/nature14539.

[27] L. Leuzzi, G. Parisi, F. Ricci-Tersenghi and J. J. Ruiz-Lorenzo. 'Dilute one-dimensional spin glasses with power law decaying interactions'. In: *Physical Review Letters* 101.10 (2008), pp. 1–4. ISSN: 00319007. DOI: 10.1103/PhysRevLett.101.107203.

[28] Yoav Levine, Or Sharir, Nadav Cohen and Amnon Shashua. 'Quantum Entanglement in Deep Learning Architectures'. In: *Physical Review Letters* 122.6 (2019), pp. 1–9. ISSN: 10797114. DOI: 10.1103/PhysRevLett.122.065301.

[29] Shuo Hui Li and Lei Wang. 'Neural Network Renormalization Group'. In: *Physical Review Letters* 121.26 (2018), pp. 1–10. ISSN: 10797114. DOI: 10.1103/PhysRevLett.121.260601.

[30] Zhaocheng Liu, Sean P. Rodrigues and Wenshan Cai. 'Simulating the ising model with a deep convolutional generative adversarial network'. In: *arXiv* (2017), pp. 1–6. ISSN: 23318422. URL: https://arxiv.org/abs/1710.04987.

[31] Erik Luijten. 'Introduction to Cluster Monte Carlo Algorithms'. In: *Computer Simulations in Condensed Matter Systems: From Materials to Chemical Biology Volume 1*. Vol. 703. March. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 13–38. ISBN: 978-3-540-35270-9.

[32] Enzo Marinari. 'Optimized monte carlo methods'. In: *Advances in Computer Simulation* (2008), pp. 50–81. DOI: 10.1007/bfb0105459.

[33] Andriy Mnih and Karol Gregor. 'Neural variational inference and learning in belief networks'. In: *31st International Conference on Machine Learning, ICML 2014* 5 (2014), pp. 3800–3809. URL: https://arxiv.org/abs/1402.0030.

[34] Alan Morningstar and Roger G. Melko. 'Deep learning the ising model near criticality'. In: *Journal of Machine Learning Research* 18 (2018), pp. 1–17. ISSN: 15337928. URL: https://arxiv.org/abs/1708.04622.

[35] J. a. Mydosh. *Spin Glasses: An Experimental Introduction*. 1993. ISBN: 0748400389. DOI: 10.1201/9781482295191.

[36] J. F. Nagle and J. C. Bonner. 'Numerical studies of the Ising chain with long-range ferromagnetic interactions'. In: *Journal of Physics C: Solid State Physics* 3.2 (1970), pp. 352–366. ISSN: 00223719. DOI: 10.1088/0022-3719/3/2/017.

[37] M. E. J. Newman and G. T. Barkema. 'Monte Carlo Methods in Statistical Physics: Chapters 1-4'. In: *Monte Carlo Methods in Statistical Physics*. Oxford University Press, 1999, pp. 1–130. ISBN: 0198517971.

[38] Kim A. Nicoli, Shinichi Nakajima, Nils Strodthoff, Wojciech Samek, Klaus Robert Müller and Pan Kessel. 'Asymptotically unbiased estimation of physical observables with neural samplers'. In: *Physical Review E* 101.2 (2020). ISSN: 24700053. DOI: 10.1103/PhysRevE.101.023304.

[39] Adam Paszke et al. 'PyTorch: An imperative style, high-performance deep learning library'. In: *arXiv* NeurIPS (2019), pp. 1–12. ISSN: 23318422. URL: https://arxiv.org/abs/1912.01703.

[40] N. Read. 'Short-range ising spin glasses: The metastate interpretation of replica symmetry breaking'. In: *Physical Review E - Statistical, Nonlinear, and Soft Matter Physics* 90.3 (2014), pp. 1–32. ISSN: 15502376. DOI: 10.1103/PhysRevE.90.032142.

[41] Philip Richerme, Zhe Xuan Gong, Aaron Lee, Crystal Senko, Jacob Smith, Michael Foss-Feig, Spyridon Michalakis, Alexey V. Gorshkov and Christopher Monroe. 'Non-local propagation of correlations in quantum systems with long-range interactions'. In: *Nature* 511.7508 (2014), pp. 198–201. ISSN: 14764687. DOI: 10.1038/nature13450.

[42] Christian P. Robert, Víctor Elvira, Nick Tawn and Changye Wu. 'Accelerating MCMC algorithms'. In: *arXiv* (2018), pp. 1–22. ISSN: 23318422. URL: https://arxiv.org/abs/1804.02719.

[43] Christopher Roth. 'Iterative retraining of quantum spin models using recurrent neural networks'. In: *arXiv* (2020), pp. 1–7. ISSN: 23318422. URL: https://arxiv.org/abs/2003.06228.

[44] J. Sak. 'Recursion relations and fixed points for ferromagnets with long-range interactions'. In: *Physical Review B* 8.1 (1973), pp. 281–285. ISSN: 01631829. DOI: 10.1103/PhysRevB.8.281.

[45] Or Sharir, Yoav Levine, Noam Wies, Giuseppe Carleo and Amnon Shashua. 'Deep Autoregressive Models for the Efficient Variational Simulation of Many-Body Quantum Systems'. In: *Physical Review Letters* 124.2 (2020). ISSN: 10797114. DOI: 10.1103/PhysRevLett.124.020503.

[46] Kyle Sprague, Juan Carrasquilla, Steve Whitelam and Isaac Tamblyn. 'Watch and learn - A generalized approach for transferrable learning in deep neural networks via physical principles'. In: *arXiv* (2020), pp. 1–6. ISSN: 23318422. DOI: 10.1088/2632-2153/abc81b.

[47] Jos Thijssen. *Computational physics, second edition*. Cambridge University Press, 2007, pp. 1–620. ISBN: 978-0-521-83346-2.

[48] Giacomo Torlai and Roger G. Melko. 'Learning thermodynamics with Boltzmann machines'. In: *Physical Review B* 94.16 (2016), pp. 1–8. ISSN: 24699969. DOI: 10.1103/PhysRevB.94.165134.

[49] Benigno Uria, Marc Alexandre Cote, Karol Gregor, Iain Murray and Hugo Larochelle. 'Neural autoregressive distribution estimation'. In: *Journal of Machine Learning Research* 17 (2016), pp. 1–37. ISSN: 15337928. URL: https://arxiv.org/abs/1605.02226.

[50] K. Uzelac and Z. Glumac. 'Finite-range scaling for a one-dimensional system with long-range interactions'. In: *Journal of Physics A: Mathematical and General* 21.7 (1988), pp. 421–425. ISSN: 03054470. DOI: 10.1088/0305-4470/21/7/011.

[51] Lei Wang. 'Generative Models for Physicists'. In: (2018), pp. 1–34. URL: https://wangleiphy.github.io/lectures/PILtutorial.pdf.

[52] Lingxiao Wang, Yin Jiang, Lianyi He and Kai Zhou. 'Continuous-mixture Autoregressive Networks for efficient variational calculation of many-body systems'. In: *arXiv* (2020), pp. 1–6. URL: http://arxiv.org/abs/2005.04857.

[53] Matthew Wittmann and A. P. Young. 'The connection between statics and dynamics of spin glasses'. In: *Journal of Statistical Mechanics: Theory and Experiment* 2016.1 (2016), pp. 1–5. ISSN: 17425468. DOI: 10.1088/1742-5468/2016/01/013301.

[54] Ulli Wolff. 'Collective Monte Carlo Updating for Spin Systems'. In: *Physical Review Letters* 62.4 (1989), pp. 361–364. ISSN: 00319007. DOI: 10.1103/PhysRevLett.62.361.

[55] Dian Wu, Lei Wang and Pan Zhang. 'Solving Statistical Mechanics Using Variational Autoregressive Networks'. In: *Physical Review Letters* 122.8 (2019), pp. 1–11. ISSN: 10797114. DOI: 10.1103/PhysRevLett.122.080602.

[56] Li Yang, Zhaoqi Leng, Guangyuan Yu, Ankit Patel, Wen Jun Hu and Han Pu. 'Deep learning-enhanced variational Monte Carlo method for quantum many-body physics'. In: *Physical Review Research* 2.1 (2020), pp. 1–8. ISSN: 23318422. DOI: 10.1103/physrevresearch.2.012039.

[57] A. P. Young. 'Monte Carlo studies of the one-dimensional Ising spin glass with power-law interactions'. In: *Physical Review B - Condensed Matter and Materials Physics* 67.13 (2003), pp. 1–8. ISSN: 1550235X. DOI: 10.1103/PhysRevB.67.134410.

[58] J. M. kosterlitz and D. J. Thouless. 'Ordering, metastability and phase transitions in two-dimensional systems'. In: *Journal of Physics C: Solid State Physics* 6.7 (1973), pp. 1181–1203. ISSN: 00223719. DOI: 10.1088/0022-3719/6/7/010.

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Happy users of `classicthesis` usually send a real postcard to the author, a collection of postcards received so far is featured here:

http://postcards.miede.de/

Final Version as of 7th June 2021 (`classicthesis` version 4.2).