Numerical exploration of statistical physics

Bukva, A.

1

Introduction

One of the recurring motifs throughout this thesis is how recent advances in computer hardware and reduced cost of computations helped us push the boundaries of physical knowledge. Our exploration starts with one of the most widely used numerical tools in any computational discipline, the Monte Carlo (MC) method. Because of the use of random numbers, these methods are believed to carry the name after the most famous casino in the world, Monte Carlo Casino in Monaco. The term *Monte Carlo* was first used in modern literature by S. Ulam, E. Fermi, J. von Neumann, and N. Metropolis while they were working on the Manhattan Project in Los Alamos National Laboratory during the second world war. MC methods will also play a significant role in this thesis as one of the primary tools for analyzing physical systems. Our journey will start with an exploration of different realizations of gauge lattice models; we will introduce simple Wegner Ising gauge theory, the fundamental block, and later build upon it by adding matter fields to the lattice. MC played a significant role in the early days of lattice gauge theories helping us advance our understanding of the theory of elementary particles. We will use it similarly to explore phase diagrams and phase transitions in our lattice gauge models. On the next stop, we will explore how closed, unitary quantum systems can (appear to) thermalize, a question that has been at the heart of statistical physics for a long time. The final stop will be one governed by the recent advances of machine learning in areas of physics. We will try to quantify the multipartite entanglement in quantum systems by combining MC simulations and newly proposed variational wave functions in the form of restricted Boltzmann machines (RBM). Some of the problems, or quirks, of MC methods can also appear in the modern training of neural networks. Most prominent is the choice of initial distribution for parameters we are trying to optimize.

Using statistical physics and field theory tools, we will find optimal initial conditions to improve neural networks' training speed and accuracy.

1.1. Statistical physics

The primary goal of statistical physics is an exploration of macroscopic quantities and the calculation thereof. Often, the systems we explore are made up of many degrees of freedom, and solving them exactly is impossible. In order to do this, we will assume that the statistical average over all possible states can replace the time average.

One of the main assumptions we make when resorting to statistical calculations instead of fully dynamically solving the system is the principle of *ergodicity*. Ergodicity states that if the system is left to evolve, all accessible states will eventually be realized. This assumption helps us often turn insolvable time integrals into relatively easy and, more importantly, simulation-friendly integrals over the probability distributions of those states. For example, let us say we want to study some volume of gas in a container. At standard temperature and pressure, one liter of oxygen contains around $3 \cdot 10^{22}$ oxygen molecules moving around the container. Just writing down equations of motion for all molecules would take a very long time, but no practical conclusion can be drawn even if we manage to do it. Hence we turn to the methods of statistical physics.

These statistical integrals are averaging over many system realizations while keeping certain parameters fixed. These sets of systems with specific parameters fixed are called *ensembles*. For example, a glass of water sitting on a counter at a fixed temperature will have many different configurations of water molecules that fluctuate while having a fixed *average* energy.

There are several well-known ensembles according to corresponding fixed quantities. If the fixed quantity is energy, the ensemble is called *microcanonical*, we can denote the total number of states with that energy as $\mathcal{N}(E)$, the probability that any of those instances are realized is $\frac{1}{\mathcal{N}(E)}$ and the probability that system is in some other energy $E' \neq E$ is simply zero.

Now we can define the *entropy* of a system as:

$$S(E) = k_B \log \mathcal{N}(E) \,. \tag{1.1}$$

Because the number of states in a system of N bits/particles is of order $\mathcal{N}(E) \sim 2^N$, it follows that the entropy is proportional to a number of particles in the system $S(E) \sim N$. We can see that this makes entropy an extensive quantity. For two non-interacting systems at energies E_1 and E_2, we get the total number of states in both systems as:

$$\mathcal{N}(E_1, E_2) = \mathcal{N}(E_1)\mathcal{N}(E_2) \,,$$

and the entropy will be:

$$S(E_1, E_2) = S(E_1) + S(E_2) \,.$$

1

Often systems are not isolated but actually in contact with some reservoir at a specific temperature T. One way we can remedy this is by including the reservoir in our calculation and using a microcanonical ensemble, but doing this results in very complex calculations and, more importantly, properties of the reservoir that are often not of interest to us. Because of this, we will introduce an ensemble at a fixed temperature called *canonical*. Now the system can be in states with the different energy that will fluctuate around some well-defined average value. The average energy of the system is defined as a sum of energies of all the states weighted with their respective probabilities:

$$\langle E \rangle = \sum_i p_i E_i \, . \tag{1.2}$$

Here p_i is a probability that the system is in the state with energy E_i. One can derive, either by entropy maximization or placing the system in contact with a heat bath, that a system with only energy as conserved quantity has a probability, also known as *Boltzmann distribution*:

$$p_i = \frac{\exp\left(-\beta E_i\right)}{\sum_i \exp\left(-\beta E_i\right)} \, . \tag{1.3}$$

We have introduced a factor, inverse temperature, $\beta = \frac{1}{k_B T}$ where k_B is Boltzman constant = $1.380649 \cdot 10^{-23} \mathrm{J} \cdot \mathrm{K}^{-1}$, and T temperature of the ensemble. Exponential factors ensure that states with energies $E_i \gg k_B T$ get suppressed while states with $E_i \leq k_B T$ have a chance of being populated. We can check that this makes sense. Let $T \to 0$, and the system is forced to a ground state energy with all higher states being suppressed. We can also define the entropy for the canonical ensemble as follows:

$$S = -k_B \sum_i p_i \log p_i \, . \tag{1.4}$$

The normalization factor is an essential quantity in Eq.(1.3). This sum over all accessible states that ensure that probability sums to 1 is called *partition function*:

$$Z = \sum_i \exp\left(-\beta E_i\right) . \tag{1.5}$$

Once we know the partition function, we can compute any quantity about the system; we have complete knowledge. Let us explore what information we can extract from the partition function. We can define the free energy at fixed temperature as:

$$F = \langle E \rangle - TS \, , \tag{1.6}$$

where T is temperature and S is entropy. We can think of free energy as energy available to the system, an intricate interplay between internal energy and entropy. At the fixed temperature, we can get free energy directly from the partition function:

$$F = -k_B T \log Z \, . \tag{1.7}$$

To connect average energy and a partition function, we can plug Eq.(1.3) into Eq.(1.2):

$$\langle E\rangle = \sum_i p_i E_i = \sum_i \frac{E_i \exp\left(-\beta E_i\right)}{Z} .$$

However, this can be rewritten in the form of a partial derivative of the partition function:

$$\langle E\rangle = \frac{\partial}{\partial \beta} \log Z . \tag{1.8}$$

As stated earlier, energy is not constant in the canonical ensemble but fluctuates. The exponential factor in Boltzmann distribution ensures that these fluctuations are around $k_B T$; we can compute them as a variance of energy:

$$\sigma_E^2 = \langle (E - \langle E\rangle)^2\rangle = \langle E^2\rangle - \langle E\rangle^2 = \frac{\partial^2}{\partial \beta^2} \log Z = -\frac{\partial \langle E\rangle}{\partial \beta} .$$

We can make a connection with specific heat that will give us a different interpretation of these fluctuations. Specific heat is a thermodynamic quantity defined as $C = \frac{\partial E}{\partial T}$ or how much the energy of a system changes as we vary the temperature. Specific heat can be measured while we keep other parameters fixed, like volume or pressure. So, for example, specific heat at fixed volume will be $C_V = \left.\frac{\partial E}{\partial T}\right|_{V=\text{const}}$. Now we can connect specific heat and the variance of energy:

$$\sigma_E^2 = k_B T^2 C_V .$$

Interesting conclusions can be drawn from here; if C_V of the system is large, it can easily absorb bigger energy fluctuations without changing its temperature. Another conclusion is that for macroscopic systems ($N \gg 1$) away from the critical point, $E \propto N$ and $C_V \propto N$, so we have that energy fluctuations scale as:

$$\frac{\sigma_E}{E} \sim \frac{1}{\sqrt{N}} . \tag{1.9}$$

In the limit, when $N \to \infty$, the energy becomes exactly $\langle E\rangle$. This is also known as a *thermodynamic limit*. These fluctuations are too small for real systems to be detected, but they are often how we calculate thermodynamic quantities in simulations.

The question now is what happens with the fluctuations of other quantities. If there are terms in the Hamiltonian $\mathscr{H}$, of form $-XY$, where Y is some field and X conjugate variable to which field Y is coupled then we have:

$$\langle X\rangle = \frac{1}{\beta Z} \frac{\partial}{\partial Y} \sum_i \exp\left(-\beta \left\langle i\middle|\mathscr{H}\middle|i\right\rangle\right) = -\frac{\partial F}{\partial Y} . \tag{1.10}$$

Now $\left\langle i\middle|\mathscr{H}\middle|i\right\rangle$ contains the term of form $-XY$ on which partial derivative acts. So if we desire to find an average value of any conjugate variable, we need to differentiate

free energy with respect to the appropriate coupled field. If there is no term coupling field to desired quantity, we can add it, find the derivative, and later set the same field to zero, restoring an original Hamiltonian. Doing further differentiation, we can see that:

$$-\frac{1}{\beta}\frac{\partial^2 F}{\partial Y^2} = \frac{1}{\beta}\frac{\partial \langle X\rangle}{\partial Y} = \langle X^2\rangle - \langle X\rangle^2 . \tag{1.11}$$

We used this variance to connect energy fluctuations to the specific heat. Now we have a general way of finding these variances from the second derivatives of free energy with respect to coupled fields. The derivative in Eq.(1.11) measures the magnitude of the response of X to changes happening in Y. This is called *susceptibility* of X to Y and is usually written as χ. This gives us a general way of calculating susceptibility directly from the simulations by measuring fluctuations in desired quantities. A practical example of susceptibility that will be important to us is the magnetic susceptibility or how total magnetization of the system changes as we vary the external magnetic field.

Some systems can go through a *phase transition*, a sudden change in a system's specific property, becoming discontinuous in the large N limit. The prominent feature of a phase transition is the appearance of a non-vanishing value of *order parameter*, some quantity that is non-zero in the ordered phase and then identically zero in the disordered phase. Different order parameters are identified in different systems; for example, in a ferromagnet, it is a spontaneous magnetization, and in the liquid-gas phase, it is the difference in the density between the liquid and gas phases at the transition line. They can be scalar or multi-component quantities as well.

The classification of phase transitions can be done by the order of discontinuous derivative of a free energy. If the first derivative of the free energy shows discontinuous behavior, we call those transitions *first order*. If, on the other hand, the first order is continuous but the second derivative is not, we label those transitions *second order*.

We can deduce some qualitative characteristics of a model using *Landau theory of phase transitions*. The main idea revolves around expanding the free energy in the powers of the order parameter. For this expansion to converge, we have to keep the order parameter small, so near the critical point. The Ising model in an external magnetic field h is a popular toy system to illustrate the main concepts. The energy of the Ising model can be expressed as:

$$E = -J\sum_{\langle ij\rangle} s_i s_j - h\sum_i s_i , \tag{1.12}$$

where spins $s_i \in \{-1, 1\}$ and we sum over all neighboring sites $\langle ij\rangle$ in a d-dimensional hyper-cubic lattice.

Let us start with the description of a second-order phase transition. Consider a gen-

eral model (like the Ising case defined above) and denote an order parameter m (in our case, total magnetization). Then we can expand the free energy as:

$$F(T,m) = F_0(T) + a(T)m^2 + b(T)m^4 + \cdots . \tag{1.13}$$

We do not have odd terms because the theory is invariant under the change $m \to -m$, which forbids odd terms in the free energy (in our case, when $h = 0$ Ising has this symmetry). The system's state can be found by extremizing the free energy, $\frac{\partial F}{\partial m} = 0$. We can immediately see that result will depend on the signs of the temperature-dependent terms. These terms can change sign with varying temperature as well. For simplicity, we will assume that $b(T) > 0$ for all T. If we do not impose this condition and let $b(T)$ have any sign, then we need to include higher terms of expansion (m^6 in our case) which can lead to the formation of tri-critical points, a phenomenon explored in the later chapters. In order to illustrate this further, we will use the free energy of the Ising model that we can calculate from mean-field theory[1]:

$$F = -\frac{1}{\beta}\log Z = \frac{1}{2}JNqm^2 - \frac{N}{\beta}\log\left(2\cosh\beta h_{\text{eff}}\right), \tag{1.14}$$

where $q = 2d$ is the number of nearest neighbors and $h_{\text{eff}} = h + Jqm$ is an effective (mean-field) magnetic field. We can expand this mean-field expression in order to compare coefficients with our free energy expansion:

$$F(T,m) = -Nk_BT\log 2 + \left(\frac{NJq}{2}(1 - Jq\beta)\right)m^2 + \left(\frac{N\beta^3J^4q^4}{12}\right)m^4 + \cdots . \tag{1.15}$$

The first term is just an additive constant, which will vanish once we take a derivative with respect to m. Consider then the quadratic term with coefficient $a(T) = \frac{NJq}{2}(1 - Jq\beta)$ there are two different cases $a(T) > 0$ and $a(T) < 0$. From the expression of $a(T) = 0$, we can also determine the critical temperature $T_c = \frac{Jq}{k_B}$. In the case of $a(T) > 0; T > T_c$, and thus when temperatures are high, we see that the only equilibrium solution is for $m = 0$. On the other hand, when $T < T_c$, for low temperature, there are three solutions, $0, \pm\sqrt{-\frac{a(T)}{2b(T)}}$. Substituting these values in Eq.(1.15), we see that the free energy of a state with $m = 0$ is higher than the other two and represents an unstable solution, Fig.(1.1). We thus find that the free energy equals:

$$F(T) = \begin{cases} F_0(T) & T > T_c \\ F_0(T) - \frac{a(T)^2}{4b(T)^2} & T < T_c \end{cases} . \tag{1.16}$$

The important point is that if $a(T)$ is a smooth function, then the equilibrium value of m also changes continuously from $m = 0$ in the $T > T_c$ regime to a $m \neq 0$ in a $T < T_c$ regime. We can also see that the free energy is continuous at the transition. However, suppose we differentiate free energy twice to get a specific heat. In that case, we will have a term $\frac{a(T)'^2}{b(T)}$ which is usually not equal to zero, and hence specific

1

heat has a discontinuous change at $T = T_c$, defining characteristics of a second order phase transition.

When we were expanding the free energy in Eq.(1.13), we said that odd terms are forbidden by the symmetry $m \to -m$, so-called $\mathbf{Z}_2$ symmetry. Then when we found the equilibrium values of m for $T < T_c$, we saw that the system had to choose one ground state of the two, $\pm\sqrt{-\frac{a(T)}{2b(T)}}$, this is the well-known phenomena of *spontaneous symmetry breaking*. The Ising model discussion above illustrates how it is tied to the second-order phase transition.

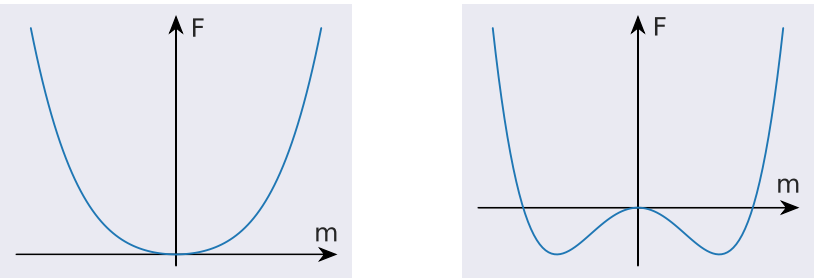


Figure 1.1: Free energy curves in the case of second-order phase transition. In the left graph, when $a(T) > 0$, we see only one equilibrium solution at $m = 0$. On the other hand, when $a(T) < 0$ (right graph), two more solutions appear. The previously stable solution at $m = 0$ now becomes unstable. We can see that the system can choose which stable solution to be in, which is what we call spontaneous symmetry breaking.

Let us briefly explore what happens when we include odd power terms of the order parameter in our expansion of the free energy F.

$$F(T,m) = F_0(T) + \alpha(T)m + a(T)m^2 + \gamma(T)m^3 + b(T)m^4 + \cdots . \tag{1.17}$$

If we look back at our definition of Ising model Eq.(1.12) we see that for $h \neq 0$ the system doesn't have $\mathbf{Z}_2$ symmetry in $m \to -m$, so we can use the expansion above:

$$F(T,m) = -Nk_BT\log 2 + \frac{JNq}{2}m^2 - \frac{N}{2k_BT}(B+Jqm)^2 + \frac{N}{24(k_BT)^3}(B+Jqm)^4 + \cdots . \tag{1.18}$$

We will use the same assumption again that $b(T) > 0$ for all T. In the regime where $T < T_c$, we again have three solutions, but now the curve is skewed, and more importantly, two minima are no longer degenerate. An introduction of the external magnetic field h broke the degeneracy that was initially there, and the other solution is called a meta-stable state. When odd terms in the expression for the free energy, $\alpha(T)$ and $\gamma(T)$, change signs, the curve goes from the left to the right picture in Fig.(1.2). This changes the true ground state, and the transition from this new meta-stable state to the true ground state with lower energy is the first-order phase transition. In the example of the Ising model, this transition occurs when the external magnetic

field h changes sign. If we increase the temperature above the critical $T > T_c$, the free energy curve looks similar to the second-order transition. However, it is shifted because of the linear term $\alpha(T)$.

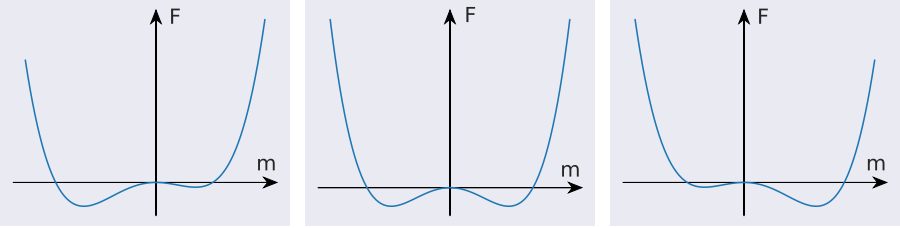


Figure 1.2: Curves of free energy in case of a first-order phase transition. Again we have three solutions, but only one is the true ground state and has lower energy than the other. A state with higher energy is a meta-stable state. Curves change shape as we change the sign of $\alpha(T)$ and $\gamma(T)$, corresponding to a phase order transition.

The toy model presented here can be solved exactly using the mean-field technique. Nevertheless, the question remains: How to compute the free energy for more complicated systems where a direct analytical approach does not work? One solution is to do it numerically or more precisely using the statistical techniques of Monte Carlo simulations that we will explore in the next chapter.

1

1.2. Monte Carlo techniques

The Monte Carlo methods are probably one of our most important numerical methods. The main idea is to obtain the approximate value of an integral by randomly sampling the value of the integrand at the expense of introducing statistical errors. Traditional grid methods that subdivide total volume become increasingly computationally expensive as we try to evaluate integrals in higher dimensions. For large systems, the number of configurations i used in Eq.(1.2) is the integral of primary interest to physicists. The expectation value of some quantity $\langle \xi \rangle$, like the total energy of a system or a magnetization, is computed by summing over all possible states weighted by their respective probabilities.

$$\langle \xi \rangle = \frac{\sum_\alpha \xi_\alpha e^{-\beta E_\alpha}}{\sum_\alpha e^{-\beta E_\alpha}} , \tag{1.19}$$

where $\beta = 1/kT$ and E_i energy of a state. Sums like these are only feasible for small systems and generally cannot be computed exactly. One possible solution that we can take is to evaluate this sum only on a small appropriate subset of states and use this as an estimate. Suppose we take L states $\{\alpha_1, \ldots, \alpha_L\}$, then our estimate will be given by:

$$\xi_L = \frac{\sum_{i=1}^{L} \xi_{\alpha_i} p_{\alpha_i}^{-1} e^{-\beta E_{\alpha_i}}}{\sum_{j=1}^{L} p_{\alpha_j}^{-1} e^{-\beta E_{\alpha_j}}} . \tag{1.20}$$

We call ξ_L the *estimator* of ξ. It is clear from the procedure that as we take more samples, the estimator becomes more accurate, and in the limit $\lim_{L\to\infty} \xi_L = \langle \xi \rangle$.

The question then is how to choose p_α. The most straightforward choice is to take all the states with an equal probability. Immediately we can see that for specific cases, this will be a terrible choice. Imagine evaluating an integral that is highly peaked around a particular value. Choosing points with equal probability will sample parts of the integral that do not contribute much toward the final result. Another way we can look at this from the physics point of view is when evaluating these sums at a very low temperature. Usually, only a handful of states will effectively contribute to the sum, so sampling over all (improbable) states will be a waste of computational time. The technique for selecting these most appropriate states is called *importance sampling*.

Return to a physics example; we know that at a specific temperature, only the states in a small energy window will have measurable contributions to the sum, while an exponential factor will suppress the others. The natural choice for the probability distribution then presents itself; we will choose states according to their Boltzmann distribution. Looking at the equation Eq.(1.20), we can see that if we pick the states according to their Boltzmann weight $p_\alpha = Z^{-1} e^{-\beta E_\alpha}$, this expression simplifies con-

siderably:

$$\xi_L = \frac{1}{L}\sum_{i=1}^{L} \xi_{\alpha_i} \,. \tag{1.21}$$

We see that all the factors have canceled out. We are left with a simple sum, or more precisely, in order to get an estimate of a quantity ξ, we are going to sample states according to their Boltzmann distribution, measure ξ in those states and then find an average over all of the measurements. When we settle on the desired distribution, the question remains how to realize it. One of the most widely used methods to achieve the desired distribution is through a *Markov process*.

In order to understand Markov processes, let us define a stochastic process at discrete times $t_1, t_2, t_3, \ldots$ and a system with a finite set of states $\alpha_1, \alpha_2, \alpha_3, \ldots$. Let us label with X_t the state of a system at time t. The defining property of a Markov process is that:

$$P(X_{t_n} = \alpha_n | X_{t_{n-1}} = \alpha_{n-1}, X_{t_{n-2}} = \alpha_{n-2}, \ldots X_{t_1} = \alpha_1) = P(X_{t_n} = \alpha_n | X_{t_{n-1}} = \alpha_{n-1}) \,, \tag{1.22}$$

or in words, Markov process is *memory-less*, the state of a system in the next time step only depends on the system's current state. Probability of generating state α_n given that system is in α_{n-1} is called *transition probability* $P(\alpha_{n-1} \to \alpha_n)$. Another condition that these probabilities should satisfy is that they do not change over time. As with all probabilities, these should satisfy the fundamental requirement:

$$\sum_{\alpha_n} P(\alpha_{n-1} \to \alpha_n) = 1 \,, \tag{1.23}$$

this means the process must generate some state, even the one the system already is in. We will end up with a *Markov chain* of states by repeatedly generating new states. Suppose transition probabilities are chosen correctly after a sufficiently long time. In that case, the chain will come to an equilibrium, and its new states will satisfy the desired Boltzmann distribution.

With chosen correctly, the following is meant. For the Markov process to achieve the desired equilibrium distribution, we must satisfy two additional conditions, *ergodicity* and *detailed balance*. Ergodicity, as we stated before, means that the Markov process should be able to reach any valid state from any valid state of the system if we let it run long enough. We know that every state in the Boltzmann distribution has a non-zero probability of appearing, so if this condition is violated, that would mean that there would be a pair of states α and γ such that if we start from a state α we would never be able to reach state γ and our goal of reaching the Boltzmann distribution would not be achievable. This does not mean that all transition probabilities should be non-zero, just that there should always be a path from two states that the process can follow.

The second condition that we need to satisfy is the one of a detailed balance. This condition ensures that the final distribution we have reached is the desired Boltzmann distribution. What this exactly means is that the process has reached an equilibrium state. Let us take not a single state α, but an ensemble with distribution p_α. Then this ensemble is in equilibrium if the rate of states transforming into any state α is the same as the rate of states transforming from α:

$$\sum_\beta p_\alpha P(\alpha \to \beta) = \sum_\beta p_\beta P(\beta \to \alpha) \,. \tag{1.24}$$

Then using Eq.(1.23) we get:

$$p_\alpha = \sum_\beta p_\beta P(\beta \to \alpha) \,. \tag{1.25}$$

If our transition probabilities satisfy this condition, the probability distribution p_α will be the equilibrium distribution. Only imposing the condition in Eq.(1.24) is not enough to guarantee that the probability distribution generated from any state will eventually settle to the desired distribution p_α. In order to understand this, let us consider transition probabilities $P(\beta \to \alpha)$. We can form a large matrix where each entry in column α and row β would be a transition probability from α to β; we call this a Markov matrix. If we label the probability that our system is in a state α at time t with $p_\alpha(t)$, then we can say that a probability that the system is in state β in a subsequent time step is:

$$p_\beta(t+1) = \sum_\alpha P(\alpha \to \beta) p_\alpha(t) \,,$$

or in a more compact, matrix notation:

$$\boldsymbol{p}(t+1) = \boldsymbol{P} \cdot \boldsymbol{p}(t) \,.$$

Now if we let this run for some time at the certain point the Markov chain will reach an equilibrium and we will have:

$$\boldsymbol{p}(\infty) = \boldsymbol{P} \cdot \boldsymbol{p}(\infty) \,.$$

which is just a form of standard eigenvector equation. However, it is also possible for the Markov process to reach *dynamic equilibrium*, which means that $\boldsymbol{p}$ rotates around several different values. Such rotation is called a *limit cycle*. If this were the case, then the eigen equation would be:

$$\boldsymbol{p}(\infty) = \boldsymbol{P}^n \cdot \boldsymbol{p}(\infty) \,,$$

where we call n the length of the limit cycle. So how can we ensure limit cycles do not happen? The easiest way is to impose the condition of *detailed balance*:

$$p_\alpha P(\alpha \to \beta) = p_\beta P(\beta \to \alpha) \,. \tag{1.26}$$

We can trivially see that if this condition is satisfied, then Eq.(1.24) is also satisfied, but more importantly, we are eliminating the possibility of cycles. In order to understand why, look at the left side of Eq.(1.26); this is an overall rate at which transition from α to β happens, while the right side is the reverse. So this tells us that, on average, the system goes from $\alpha \to \beta$ as often as $\beta \to \alpha$. This would inherently be violated in the limit cycle as the states need to go from one to the other in a cyclic way. Another way we can justify the detailed balance condition is that most physical systems satisfy it in some way. The majority of the physical systems exhibit time-reversal symmetry. If we had a cycle in such a system upon reversing the arrow of time, the cycle would also reverse its direction. However, this reversal of cycle direction would completely change the system's dynamics, and any already established equilibrium would not be the same. In order to represent physical systems without this behavior, we would also like to remove it from our models.

We have established how to generate the desired probability, and from the detailed balance, our transition probabilities should satisfy the following:

$$\frac{P(\alpha \to \beta)}{P(\beta \to \alpha)} = \frac{p_\beta}{p_\alpha} = e^{-\beta(E_\beta - E_\alpha)} . \tag{1.27}$$

Here in the last step, we substituted the Boltzmann distribution. If we choose transition probabilities that satisfy this and the ergodicity condition, our Markov process should converge to the Boltzmann distribution. The procedure described so far looks straightforward; we need to choose the correct transition probability $P(\alpha \to \beta)$, and we are set. This is not always such an easy task, and there can be many different ways of creating a state β from α that still, in the end, does not produce the desired distribution. There is a trick that can help us construct the desired process using any algorithm that we come up with, called the *acceptance ratio*.

The main observation that lies at the heart of the acceptance ratio is that we can modify the probability of a state staying at the same state as much as we want (still with the constraint that probability should be positive and less than one) and automatically satisfy Eq.(1.27). This allows us to tune our transition probabilities as we like and then, by adjusting the "staying" probability, satisfy the required condition. Let us see how we can do this. First, we will break the transition probability into two parts:

$$P(\alpha \to \beta) = g(\alpha \to \beta) A(\alpha \to \beta) .$$

The first quantity on the right side, $g(\alpha \to \beta)$, is called the *selection probability*, the probability given an initial state α that we will end up in the state β. The second quantity, $A(\alpha \to \beta)$, is the acceptance ratio, the probability that we accept the generated state. Now if we expand the Eq.(1.27) in terms of selection probability and acceptance ratio, we will get:

$$\frac{P(\alpha \to \beta)}{P(\beta \to \alpha)} = \frac{g(\alpha \to \beta) A(\alpha \to \beta)}{g(\beta \to \alpha) A(\beta \to \alpha)} . \tag{1.28}$$

From the equation above, we see that the ratio $\frac{A(\alpha\to\beta)}{A(\beta\to\alpha)}$ can take any value between 0 and ∞. This means that selection probabilities can be tuned to anything we like, and the acceptance ratio will absorb any difference. Partitioning transition probabilities into acceptance ratio and selection probability enables us to create a Monte Carlo algorithm that will generate random state ν given state μ and then adjust selection probabilities for these states so they satisfy the condition in Eq.(1.28) and reach the desired Boltzmann distribution. So we finally reached a theoretically complete algorithm to generate the desired Boltzmann distribution.

Algorithm 1 Markov process algorithm

1: design an algorithm that will generate a random new state β given that we are at α with some probabilities $g(\alpha \to \beta)$
2: accept that state with the probability $A(\alpha \to \beta)$
3: adjust acceptance ratios in such a way that we satisfy Eq.(1.28)
4: after reaching it's equilibrium state, Markov process will have the desired Boltzmann probability distribution.

The algorithm outlined above looks nice and easy, but we still need to pay attention to how we choose our acceptance ratios. If they are small, we will usually stay in the same state and not move at all. We need to make a delicate balance so that our algorithm explores the phase space without slowly crawling around it. This can be ensured by making the acceptance ratio close to one. The good thing is that Eq.(1.28) only fixes ratio $\frac{A(\alpha\to\beta)}{A(\beta\to\alpha)}$, which means that we have the freedom to multiply both of them with some constant. In practice, this often means we fix the larger to one and then multiply the other with an appropriate constant to keep the ratio fixed. Another thing that we can try to do while designing a new algorithm is to put as much of $P(\alpha \to \beta)$ into the selection probability because the perfect algorithm would be the one where the states are only selected by the transition probability, which means acceptance ratio is always one. In the absence of a perfect, we can try to keep our acceptance ratio as close to one as possible.

We have covered the theoretical part of designing an algorithm to ensure that the resulting chain will have the Boltzmann distribution. Let us look into one of the most common algorithms, *Metropolis algorithm*.

In order to show all the details of how this algorithm works, we will again use the Ising model, where "spins" s_i live on the sites of a lattice and can take values $\{-1,+1\}$. If we have d dimensional lattice of size N, our system can be in 2^{N^d} possible states. These numbers grow exponentially as we go to higher dimensions and bigger lattices. This type of problem is the perfect playground for Monte Carlo methods. To recall from

Eq.(1.12) the Ising Hamiltonian is:

$$E = -J \sum_{\langle i\, j \rangle} s_i s_j - h \sum_i s_i \,, \tag{1.29}$$

where J is the interaction energy between the nearest neighbors and h is the external magnetic field. The most common questions regard the values of magnetization m and specific heat C_V at the fixed temperature, which we computed earlier in the mean-field limit. For simplicity, we will consider the Ising model with no external magnetic field $h = 0$

As we showed in the section 1.1 in the thermal equilibrium, energies of the system stay in a tiny window around the mean value; they do not fluctuate much. We want to replicate this behavior in our algorithm as well. The easiest way we can do this is by flipping a single spin at a time. This type of algorithm has a *single spin-flip dynamics*. In the d dimensional lattice, the maximum energy difference would be $2zJ$, where z is *lattice coordination number*, or simply the number of neighbors a site has. Another advantage of flipping a single spin is that we also satisfy the ergodicity condition.

The Metropolis algorithm now sets the selection probabilities $g(\alpha \to \beta)$ equal for all possible states β while setting it to zero for all the other ones. Let us say that we have N^d spins in our system; that means there are N^d different spins that we could flip and also N^d different states that we could end up in. This means that we have N^d different selection probabilities $g(\alpha \to \beta)$ that are non-zero, and all of them have the same value:

$$g(\alpha \to \beta) = \frac{1}{N^d} \,.$$

In practice, we randomly pick a site with equal probability and flip its spin. Now our condition of the detailed balance (1.28) is:

$$\frac{P(\alpha \to \beta)}{P(\beta \to \alpha)} = \frac{g(\alpha \to \beta) A(\alpha \to \beta)}{g(\beta \to \alpha) A(\beta \to \alpha)} = \frac{A(\alpha \to \beta)}{A(\beta \to \alpha)} = e^{-\beta(E_\beta - E_\alpha)} \,. \tag{1.30}$$

If we remember the algorithm design steps, we will set the largest of the acceptance ratios to 1 and adjust the other one to make this the most efficient algorithm. Assume that state α has the lower energy than β, $E_\alpha \le E_\beta$. Then the larger acceptance ratio would be where we go from the state β to the state α, $A(\beta \to \alpha)$, we will set it to 1. In order to satisfy Eq.(1.30) we must then set $A(\alpha \to \beta) = e^{-\beta(E_\beta - E_\alpha)}$. Now our final algorithm is:

$$A(\alpha \to \beta) = \begin{cases} e^{-\beta(E_\beta - E_\alpha)} & \text{if } E_\beta - E_\alpha \ge 0 \\ 1 & \text{otherwise} \end{cases} \,. \tag{1.31}$$

Put in words, this means that if we select a state with lower energy, we always accept the move, and if we select the state with higher energy, we will accept it with the probability $e^{-\beta(E_\beta - E_\alpha)}$.

1

So far, we only said that we must wait long before our system reaches equilibrium. Now we will specify what this exactly means. First, let us define this period. The time we need to wait for the system to reach the thermal equilibrium is called the *thermalization time*, τ_T. How do we know how long is enough? We could look at the state of a system as in the Fig.(1.3) and then gauge when it has reached the equilibrium, but the better and simpler way to do this is to plot some quantity, like energy or magnetization, over "time" and see when it settles around the average value. The usual way to measure time in Monte Carlo simulations is in terms of *sweeps*. Sweep is the number of updates we do to flip all the spins on the lattice. So if we sequentially try to flip spins in the lattice after N^d attempts, we will complete one sweep. In Fig.(1.4), we see that after 300 sweeps, the average magnetization and energy have thermalized. However, in practice, this is not always enough. Sometimes if the energy landscape is particularly rough, we could end up in one of the *local minima* and oscillate around it. In order to avoid this, we can start our simulation with different initial conditions, like at $T = 0$ (all spins aligned) and $T = \infty$ (completely randomly oriented spins), and let them run. When two runs settle around the same average value, we have good reason to believe that thermalization has completed and we have reached equilibrium Fig.(1.5). If we are still worried that we might have ended up in some local minimum, we can do a third run with a different randomly oriented configuration and repeat the same procedure.

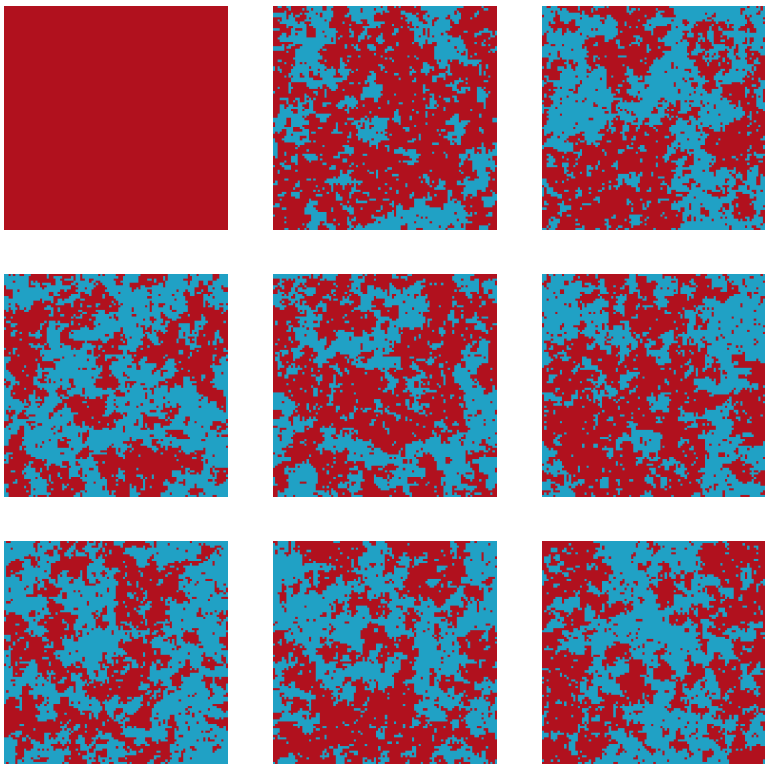


Figure 1.3: Time slices of the spin configuration on a 100 × 100 lattice throughout the simulation. Red are spins −1, and blue spins are +1. We have started at $T = 0$ configuration with all the spins aligned and then simulated $J = 1$ and $T = 2.4$ for 1000 sweeps. Slices were taken every 100 sweeps. The graph shows that the system reached equilibrium configuration around the 400th sweep.

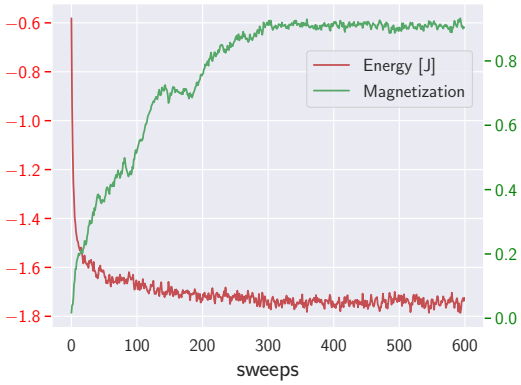


Figure 1.4: Graph of magnetization per site (green squares) and energy per site (red circles) for a 2D Ising model with no external magnetic field ($h = 0$) on a lattice $N = 100$, $J = 1$ near the critical temperature ($T_c \sim 2.26$) $T = 2.0$. We can see that energy and magnetization reach their thermal values around 300-400 sweeps.

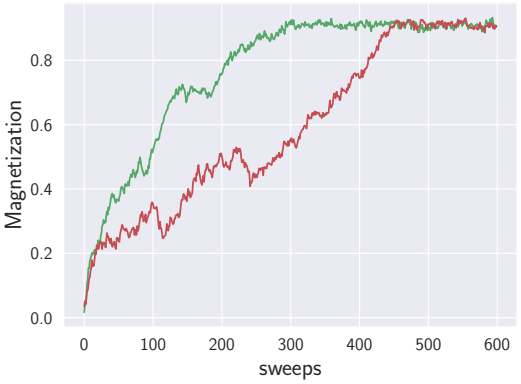


Figure 1.5: Graph of magnetization per site for two 2D Ising models with different initial configurations (at $T = \infty$), no external magnetic field ($h = 0$) on a lattice $N = 100$, $J = 1$ and equilibrium temperature $T = 2.0$. We can see that two different initial configurations had different paths through the phase space, but both have settled on the same value for the magnetization per site. Waiting for around 500 sweeps would guarantee that our Monte Carlo chain has thermalized for these values of J and T.

We have established how to get an adequately converged Monte Carlo simulation. The next step is measuring some quantity, such as energy or magnetization. If we recall our algorithm's exact dynamics, we will realize that the single flip dynamics states are actually highly correlated. This implies that two or more successive measurements will be correlated. In order to solve this, we will wait for a certain amount of sweeps between successive measurements, called *correlation time*, τ.

In order to estimate autocorrelation time, we will use *time-displaced autocorrelation*

function. We give here an example of it for magnetization:

$$\chi(t) = \int \mathrm{d}t'(m(t')m(t'+t) - \langle m\rangle^2) \,. \tag{1.32}$$

This is just the two-point function we have seen in the first part of the introduction, but now instead of being between two sites i and j, it is between two time-steps t and t'. One assumption that we have to make in order to make this work is that the autocorrelation function falls off exponentially at long times (which is a very reasonable assumption to make):

$$\chi(t) = e^{-\frac{t}{\tau}} \,. \tag{1.33}$$

We can see from this that waiting time τ between two measurements will only reduce the autocorrelation function by $1/e$, so the most common practice is to take measurements every 2τ sweeps. Given the exponential form of the autocorrelation function Eq.(1.33) we can define what the *integrated correlation time*:

$$\int_0^\infty \frac{\chi(t)}{\chi(0)}\mathrm{d}t = \int_0^\infty e^{-\frac{t}{\tau}}\mathrm{d}t = \tau \,. \tag{1.34}$$

This equation gives us a direct way of estimating the autocorrelation time. To complete the procedure, we need a discrete version of the Eq.(1.32) that we can use in order to estimate the autocorrelation time:

$$\chi(t) = \frac{1}{t_{\max}-t}\sum_{t'=0}^{t_{\max}-t} m(t')m(t'+t) - \left(\frac{1}{t_{\max}-t}\right)^2 \sum_{t'=0}^{t_{\max}-t} m(t') \sum_{t''=0}^{t_{\max}-t} m(t''+t) \,. \tag{1.35}$$

When using this equation to calculate χ, we have to take care that at long times when t gets close to $t_{\max}$, the upper limit in the sums gets very small, and because of the statistical nature of the measurements, errors can get very large. In order to avoid this, we can run a simulation long enough (hopefully for several autocorrelation times), and then we would not need to worry about the long tails of $\chi(t)$.

We have established how to generate new states from the old one, measure quantities of interest and calculate autocorrelation time so our sample end up uncorrelated. The final step is to quantify the errors of those measurements in order to get the complete results. Our errors can be divided into two categories, *systematic errors* and *statistical errors*. We have introduced systematic errors due to the procedures we used to measure observables. Another source of systematic errors can be if we do not wait long enough for the system to thermalize. Also, if we do not collect enough samples after the thermalization and let our samples be correlated. Systematic errors are hard to measure, and we can mitigate them by following the correct procedure for reaching the equilibrium and collecting enough uncorrelated samples after. On the other hand, statistical errors are more easily measured; they arise from the random nature of the Monte Carlo simulations and the system's thermal fluctuations. The simplest way to minimize statistical errors is to collect more uncorrelated samples.

1

Statistical errors of a quantity are relatively easy to estimate. As we stated, due to the thermal fluctuations from one state to the other, measurements (let us take magnetization m) vary around some average value. The natural way is to take the mean of all of these samples as the actual average value and the error of the mean as an error to that estimate. Let us take N measurements of magnetization m; our best estimator for the true average is the mean value:

$$\langle m \rangle = \frac{1}{N} \sum_{i=0}^{N} m_i \,, \tag{1.36}$$

and the estimate of the standard deviation of the mean is:

$$\sigma^2 = \frac{1}{N(N-1)} \sum_{i=0}^{N} (m_i - \langle m \rangle)^2 = \frac{1}{N-1} \left(\left\langle m^2 \right\rangle - \langle m \rangle^2 \right) . \tag{1.37}$$

This might look similar to the susceptibility we defined in section 1.1. There we calculated susceptibility as a variance of the energy over the given ensemble of states, while here, the σ^2 is an unbiased estimator for the true variance of the mean. An essential fact about Eq.(1.37) is that it assumes that our samples are statistically independent. If, on the other hand, we have samples that were sampled every Δt sweeps and the autocorrelation time is τ, then our estimate for the standard deviation is[2]:

$$\sigma^2 = \frac{1 + \frac{2\tau}{\Delta t}}{N-1} \left(\left\langle m^2 \right\rangle - \langle m \rangle^2 \right) . \tag{1.38}$$

We can immediately see that if $\Delta t \gg \tau$, this expression would reduce to the Eq.(1.37), but often, because of practical reasons, we have $\Delta t \ll \tau$. If that is the case, then we can practically ignore the 1 and note that the number of samples is the total amount of time divided by the sampling interval:

$$N = \frac{t_{\max}}{\Delta t} \,.$$

For the large number of samples we have:

$$\sigma^2 = \frac{2\tau}{t_{\max}} \left(\left\langle m^2 \right\rangle - \langle m \rangle^2 \right) . \tag{1.39}$$

A neat property of this estimation is that our standard deviation does not depend on Δt, which means we can sample with any frequency we want.

In some cases, it is not feasible to calculate the errors this way, for example, in any derived quantities that depend on the average values, like specific heat. We only have access to the average values after we finish running and do not have access to specific heat during every time step. In principle, we could do a detailed analysis of error propagation taking into account how $\langle E \rangle$ and $\left\langle E^2 \right\rangle$ errors are correlated, but this would be a complex and error-prone process, also the resulting equation would

be highly dependent on the initial equation we used to calculate the observable. This would mean that every time we have a different form of an equation, we would have to do the same analysis again. Luckily, we can use robust methods to estimate an error of these quantities. The most known and well-established is *bootstrap method*.

Bootstrap belongs to a class of the *resampling methods*. Let us first sample N measurements of the energy. Next, we can consider these N measurements as a state space for the energy, forming some probability distribution. In essence, we want to infer the information about this probability and functions of it. For energy, our goal would be to calculate the mean and the error. The idea is to draw N new samples with equal probability and repetition from the original set of N samples, repeating the procedure M times. Calculating the mean for each set of M repetitions would get us M values of the mean. These new M values represent the probability distribution of the mean, and we can use them to calculate the standard deviation. This method is advantageous when, for example, we have stored some observable values but want to estimate some function of it.

An example of a function of an observable would be the specific heat $C_V(\langle E\rangle, \left\langle E^2 \right\rangle)$. The procedure would go like this, from initial N measurements sample new N samples with repetition M times. For each of the M samples, we compute C_V, leaving us with a list of specific heats that we call a. Now it turns out that the standard deviation of the specific heat is given by[2]:

$$\sigma^2 = \langle a^2\rangle - \langle a\rangle^2 \,. \tag{1.40}$$

This means we take the standard deviation of the newly constructed list of C_V values, and the mean would simply be the average of a. It can also be shown that even if we draw correlated samples, bootstrap will estimate standard deviation equally well.

So far, we have covered the basic steps and algorithms for constructing the Markov process with the desired distribution. These steps are the same for multiple fields where Monte Carlo simulation might be used. Now we focus on the applications in physics. We will stick with the $2D$ Ising model. At the mean-field level, Ising in $2D$ dimensions goes through the phase transition as we change the temperature. We can think of it this way, at a very high temperature $T = \infty$, all the spins are randomly oriented, with plenty of energy at their disposal and no order among the spins. This is a paramagnetic phase, where the total magnetization of the system is zero.

As we decrease the energy, spins try to align more and more, as this is now energetically more favorable, and at a certain point, the total magnetization becomes finite. This phase, where the total system has non-zero magnetization, is the ferromagnetic phase. As discussed in the previous section, this phase transition belongs to the second-order phase transitions class. Second order means the second derivative of the free energy is discontinuous. The standard second derivative is specific heat

1

$C_V = -T\left(\frac{\partial^2 F}{\partial T^2}\right)\Big|_{V=\text{const}}$. The straightforward way to see if the system goes through a phase transition is to plot specific heat as a function of the changing variable. Thus, specific heat should be diverging at the transition point for the second-order phase transition. Strictly speaking, diverging behavior is only seen in the thermodynamic limit (for infinite systems). Since we can only simulate systems of finite sizes, we will see that specific heat will develop a sharper peak as we increase the size. This can be seen in Fig.(1.6). We can compare the value of T with the exact known critical temperature $T_c = \frac{2J}{\ln\left(1+\sqrt{2}\right)} \approx 2.27J$, as well as the meanfield result $T_c = 4J$.

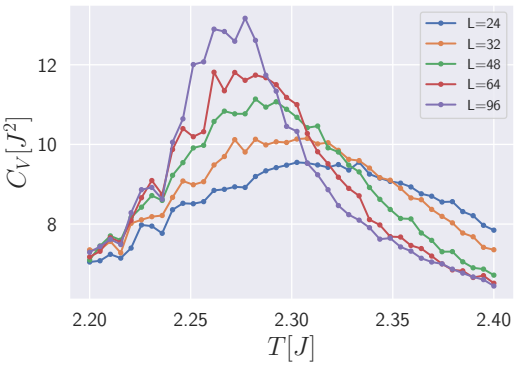


Figure 1.6: Specific heat curves C_V of 2D Ising for $J = 1$ on a square $L \times L$ lattice around the transition point for different system sizes. Strictly speaking, specific heat only shows diverging behavior for infinite system sizes. Because of this, determining the exact point of transition through these curves is not very precise. The exact value of critical temperature is $T_c = \approx 2.27$, while the mean-field result is $T_c = 4$ (way off on the above graph).

Now that we have identified the existence of a continuous phase transition, we would like to describe it in more detail for two reasons: both related to the numerical implementations. The first is critical slowing down, and the second is finite size determination of free phase transition. Landau's theory of phase transitions shows that the free energy has a singularity in the thermodynamic limit, usually described by a power law of the observables near the critical point. Underlying this divergence is a divergent *correlation length* ξ. It gives a sense of how correlated or ordered the spins in our system are, and it is diverging near the transition according to the:

$$\xi \sim |T - T_c|^{-\nu}\,, \tag{1.41}$$

whit ν the *critical exponent*. The specific heat has a similar shape near the criticality:

$$C_V \sim |T - T_c|^{-\alpha}\,, \tag{1.42}$$

with the difference that α can now be positive and negative. From the renormalization group theory, we know that these critical exponents are related by scaling

relations. In most cases, only two exponents are independent, and the rest can be derived from them[2]. These two exponents fully characterize the critical behavior of the model. Also, it can be shown that models often have the same set of these critical exponents and can be further categorized in *universality* classes. This means that models with different microscopic details exhibit the same behavior near the critical point, and details get washed away. Everything is controlled by ν.

The issue is that these divergences only occur in the thermodynamic limit. For finite-size systems, the critical behavior is smeared out, and estimating the location of the peak will introduce the error. Due to a related problem, critical slowing down, one cannot simply sample more[2].

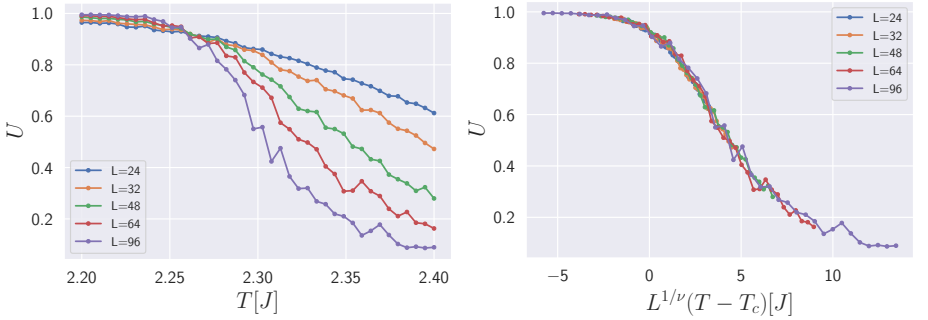
One better way to determine the transition point and critical exponents is through the technique of *finite size scaling*. Using the results from the renormalization group, we know that the magnetization scale, close to the transition, with system size L as:

$$\langle m_L \rangle \sim L^{\frac{\beta}{\nu}} \tilde{M}\left[L^{\frac{1}{\nu}}(T - T_c)\right] , \tag{1.43}$$

where $\tilde{M}(x)$ in unknown scaling function. We can see from the Eq.(1.43) that $\langle m_L \rangle / L^{\beta/\nu}$ at $T = T_c$ will be independent of the system size, as the argument of $\tilde{M}$ will be zero. So this means that scaled magnetizations will all have the same value precisely at the critical temperature. On the plot where these different scaled magnetization curves would be drawn together at the point of critical temperature, they would intersect each other. If we want to be completely precise, we are ignoring some non-analytic corrections to the scaling so the curves intersect only at the same point in the sense $L \to \infty$. Doing finite-size scaling using this method is not helpful in practice because neither β nor γ is known a priori. A better approach is to compute a combined quantity that is dimensionless. One such quantity is the *Binder cumulant*[3]:

$$U = \frac{1}{2}\left[3 - \frac{\langle m^4 \rangle}{\langle m^2 \rangle^2}\right] \sim \tilde{G}\left[L^{\frac{1}{\nu}}(T - T_c)\right] . \tag{1.44}$$

In the limit, $T \to 0$, Binder cumulant goes to 1; in the opposite limit, when the temperature $T \to \infty$ Binder goes to 0. Because Binder cumulant is a dimensionless quantity, different L curves should approximately intersect at the same point, given that the non-analytic corrections are small. We can use this to determine ν, see Fig.(1.7a). On the other hand, if we plot all the curves with the appropriate critical scaling ν on the plot U vs. $L^{1/\nu}(T - T_c)$ they all fall on top of each other as in Fig.(1.7b).



(a) (b)

Figure 1.7: **(a)** Binder cumulant curves for different system sizes. Because the Binder cumulant is a dimensionless quantity, all the curves cross at the same point, given that the non-analytic corrections are small. This method is more robust and reliable for determining the exact critical temperature $T_c \sim 2.26$. **(b)** After doing the finite size scaling for the Binder cumulant data in Fig.(1.7a), all the curves have collapsed on a single line. We can use Binder cumulant curves to determine the exact position of T_c and then use that to fit all the curves to determine the critical exponent $\nu = 1$.

In this thesis, we shall use the Binder cumulant differently to determine the order of a phase transition. In the case of continuous, second-order phase transition, the Binder cumulant has a smooth transition. In the case of a first-order phase transition, instead of being continuous, the Binder cumulant has a dip at the point of the critical transition. The dip occurs because, at the point of transition order, the parameter experiences a discontinuity which is also present in any functions that depend on it. This dip diverges with the system size[3]. We can use Binder cumulant to see whether the actual phase order exists and determine its order. Another possibility is the case of a weak first-order transition, where the Binder cumulant still exhibits a dip, but this dip is not diverging with the system size.

1.3. Machine learning

Since discovering quantum mechanics, physicists have sought better, more accurate approximation techniques to solve real-world quantum problems. As we know from the introductory quantum mechanics courses, only a handful of toy models Schrödinger equation can be solved exactly. If we start considering systems with many interacting particles, our analytical methods can only serve up to a certain point. Some well-known numerical techniques, such as exact diagonalization and tensor networks, also start having problems when we apply them to systems of many highly interacting particles. The Monte Carlo techniques reviewed in the previous section can provide approximate answers. Recently, one of the pioneering methods that used machine learning was introduced by Carleo and Troyer [4], where they consider an RBM network as an ansatz in variational quantum Monte Carlo.

The term *machine learning* (ML) has become an everyday expression in recent times. It can be defined as a sub-field of Artificial intelligence primarily tasked with developing algorithms for learning from experience (or data). The backbone of ML is based on applied statistics while also drawing some ideas from statistical physics. The main goal of ML is to estimate unknown, often very complicated, functions that depend on a large number of unknowns and give useful predictions on new (unknown) data. This program of pushing ever bigger and more complex algorithms led to the development of *deep learning* (DL), where massive networks with millions of parameters are trained on more extensive data sets. The development of highly efficient and parallel GPUs led to a boom in developing new ML and DL algorithms. The main appeal of ML and DL is that some tasks, hard to put in an algorithmic form, can now be tackled with ease. A typical example would be recognizing a cat, dog, or human being in a picture or detecting a new phase state of a quantum matter. ML and DL have grown tremendously in recent years; hence a complete introduction is simply impossible here, and we will try to introduce the essential elements of both fields used in the later chapters.

Before actually doing any computation, we need to establish the main ingredients we need to make the computer learn [5]:

- a *task* that we need to solve, like regression, classification, generating pictures, or learning how to play a game
- *data* in the case of ML and DL, we can look at the data as an experience that our algorithm lives and learns from. Some forms of data can be given in a table, like pictures of handwritten digits and their labels, or it can be given in the form of a reward, for example, how long the robot managed to walk
- a *model* that learns from the data.

Usually, more than these are needed to specify the learning algorithm fully. We need to define a measure that will compare the performance of our algorithm with the actual known data. Our task is to minimize this difference between the predicted and known data.

- *Regression* can be viewed as one of the most common forms of ML. We assume that there is some (potentially unknown) relation between two variables $\mathbf{x}$ and $\mathbf{y}$, and we try to learn what that relation is. We will call variable $\mathbf{x}$ an input and variable $\mathbf{y}$ an output. These variables are not restricted to a single dimension; they are often multi-dimensional. As stated, our task is to find some function f that will satisfy the relation $\mathbf{y} = f(\mathbf{x})$. The simplest example would be a linear regression, where we assume that our function has the form $\mathbf{y} = \mathbf{A}\mathbf{x} + \mathbf{b}$, and we have free parameters $\mathbf{A}$ and $\mathbf{b}$ to optimize.

- *Classification* is a task where our output variables do not have a continuous output but are rather discrete and categorized in different *classes (labels)*. In comparison with the regression, we are now trying to find a function that will map our input $\mathbf{x}$ to a target $\mathbf{y}$ but also encode a representation for multiple different classes. The simplest example would be a binary classification, a case when we need to distinguish between two different labels, for example, is this a picture of a cat or not. One of the most famous datasets that we will use later is MNIST (Modified National Institute of Standards and Technology), which consists of 70.000 handwritten digits distributed over 10 "classes", the digits $0, 1, 2, \ldots, 9$.

The data, the main ingredient in many machine learning tasks, comes in datasets $\mathscr{D}$, containing data points $\mathbf{x}_i$, $\mathscr{D} = \{\mathbf{x}_i\}$. Depending on the information available in our data set, we can divide the types of learning we can do:

- *Supervised learning* is a class of problems where our data points are *labeled*. It can either be a regression or a classification. Methods used to do supervised learning can be classical ML or more modern and complex DL.

- *Unsupervised learning* is applied when data does not have accompanying labels that we can use. Some examples of unsupervised learning methods are the initial pre-processing of data when we are trying to select the best features to use, for trying to find some order in the given data set by grouping points in clusters according to some features or increasing the dimensionality by adding features through generative models. An example from physics application is trying to distinguish two phases of a matter when we do not understand the underlying process governing it.

- *Reinforcement learning*, in this case, we usually do not have a data set, but rather some environment that our model explores. Through the actions of a model and feedback received from the environment model chooses the next best step in order to maximize some in advance predetermined metric.

In this thesis, we will explore some topics from the supervised classification problems through the DL and reinforcement learning methods to find the ground state of a quantum system.

The last ingredient we need is a model. In general, this is some function $f_{\boldsymbol{\theta}}(\mathbf{x})$ of the input data. There are various forms of functions, each suitable for a specific job. The model function is specified by giving its mathematical form, in ML the network, and a set of variable parameters $\boldsymbol{\theta}$, in ML the weights. Our job then is to train a model to find an optimal set of parameters $\hat{\boldsymbol{\theta}}$ that will minimize some target loss function $\mathscr{L}$ or maximize a model performance. In essence, we are modeling the true relation between $\mathbf{x}$ and ansatz $f_{\boldsymbol{\theta}}(\mathbf{x})$. In physics, we know this from variational wave function approximation. We will review this later. One of the most common loss functions to minimize is the mean-squared error (MSE) used in regression problems:

$$\mathscr{L}_{\text{MSE}} = \sum_{i=1}^{N} (y_i - f_{\boldsymbol{\theta}}(x_i))^2 \,, \tag{1.45}$$

where N is the number of data inputs. For classification problems, the most widely used loss function is a cross-entropy (CE). The simplest form of CE is when we only have two categories, binary cross-entropy (BCE). In the case of multiple different categories, we use categorical cross-entropy (CCE):

$$\mathscr{L}_{\text{BCE}} = -\sum_{i=1}^{N} y_i \log\big(f_{\boldsymbol{\theta}}(x_i)\big) + (1 - y_i) \log\big(1 - f_{\boldsymbol{\theta}}(x_i)\big) \,, \tag{1.46}$$

$$\mathscr{L}_{\text{CCE}} = -\sum_{i=1}^{N} \sum_{j=1}^{K} y_{i,j} \log\big(f_{\boldsymbol{\theta}}(x_i)\big) \,, \tag{1.47}$$

where now index j goes over K different categories and the constraint $\sum_i f_{\boldsymbol{\theta}}(x_i) = 1$. The last formula is written in a form where labels $y_{i,j}$ are given in a *one-hot encoding*.

$$y_{i,j} = \begin{cases} 1, & \text{if } y_i = j \\ 0, & \text{otherwise} \end{cases} \tag{1.48}$$

For example, if we have 5 categories, label $y_3 = (0,0,1,0,0)$ is a vector with all zeros and one at the i-th (3rd) class position.

We need to specify the minimization procedure after choosing a model and loss function. This procedure can either be gradient-based or gradient-free. A standard widely used is *gradient descent* (GD). In short, we start our learning process by initializing

a random set of parameters $\boldsymbol{\theta}_0$ and compute the loss function. After that, we compute the gradient of a loss function with respect to the model parameters. Finally, we update our parameters by subtracting previously computed gradients along the steepest direction:

$$\theta_{j+1} = \theta_j - \eta \frac{\partial \mathscr{L}}{\partial \theta_j} , \tag{1.49}$$

where η is a learning rate, it controls the size of steps we take. Every time we complete one full update, we say that one *epoch* has passed. Choosing an appropriate learning rate is a delicate process; if the learning rate is too small, our training will take forever, and we will waste computing resources. If the learning rate is too big, our algorithm might never converge. The most common way of choosing a learning rate is through trial and error.

Gradient descent is only sometimes the best algorithm to use. The algorithm will converge if the learning rate is small enough, but there is no guarantee that the minimum reached is the actual global minimum, not a local one. A clever way to deal with this problem is to introduce some stochasticity. There are various ways to do so, but one way is to incorporate it directly into GD. A standard, modified version of GD is called stochastic gradient descent (SGD). One computes the gradients not on a whole input data set but only on a small set of *batches*. The initial data set is divided into batches of equal size, and in each step, the neural network is trained only on a single batch. We say that one epoch has passed after we trained the neural network on all the batches and went through the data set once. Doing this has two benefits; one is that now we do not need to compute gradients with respect to all the inputs hence reducing the computational cost, and second by only updating parameters after computing gradients with respect to one batch, we are introducing randomness and avoiding saddle points and narrow local minima.

A further improvement to an SGD is in the form of momentum or inertia. This serves as a memory of the direction in which we are moving and helps us move in a direction with consistent but small gradients while avoiding oscillations in high curvature directions:

$$v_j = \gamma v_{j-1} + \eta \frac{\partial \mathscr{L}}{\partial \theta_j} , \tag{1.50}$$

$$\theta_{j+1} = \theta_j - v_j , \tag{1.51}$$

where v_j is a running average of the previously computed gradients and $(1-\gamma)^{-1}$ sets a time scale of how far back we want to look. If we set $\gamma = 0$, we revert to a previous case of (S)GD without momentum.

Parameters like η, γ, the number of batches we use, and the total number of epochs are called *hyperparameters*. Besides these, all parameters that control the learning process can be put in this category. In order to find a good set of hyperparameters,

the established method is to split the input data into two sets, *training set* and *validation set*. Doing this is a good practice besides finding an optimal set of hyperparameters. We want an independent data set that was not seen before, on which we will evaluate our model. In practice, we might have several choices for our models, and a consistent way to compare them is to see how they perform on previously unseen data. Being able to make generalizations and predict results from unknown data is the primary goal of ML and DL. We can get in a situation whereby choosing a very complex model will fit every possible feature in our data set (*overfitting* the data), but then when presented with a new, previously unseen data model, will perform terribly.

On the other hand, if we choose a simpler model, our loss might be higher for the training data. However, when presented with new data, this simpler model will outperform a more complicated one, see Fig.(1.8). This is called *bias-variance trade-off*.

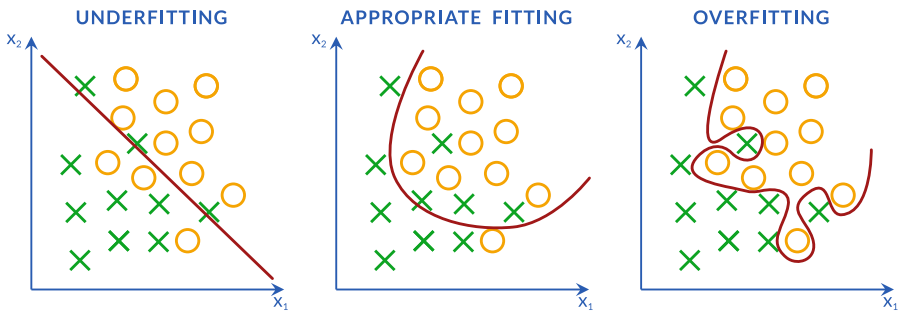


Figure 1.8: Three different scenarios of model selection. On the left, the selected model does not have enough representational power to capture desired features in our data set. The center image model is complex enough to balance bias and variance while still being able to generalize on new data. We have a too-complex model for the given data set on the right. Too many parameters and the model will be able to capture all the features of the data, even the noise, leading to an inferior generalization.

This principle can be nicely illustrated on an example of the MSE loss function and linear relation between input and target:

$$y = f_{\boldsymbol{\theta}}(\mathbf{x}) + \epsilon \,, \tag{1.52}$$

where ϵ is a noise distributed according to a Gaussian distribution $\epsilon \sim \mathcal{N}(0, \sigma_\epsilon^2)$. In this case, the MSE loss function would be:

$$\mathscr{L}_{\text{MSE}} = \sum_{i=1}^{N} (y_i - f(\mathbf{x_i}; \boldsymbol{\theta}))^2 \,. \tag{1.53}$$

Following the simple exercise in algebra and probability [5], we can factor this loss into three individual components:

1

$$\text{error} = \text{bias}^2 + \text{var} + \text{noise}\,, \tag{1.54}$$

where *bias* is:

$$\text{bias}^2 = \sum_{i=1}^{N} \left(f(\mathbf{x}_i) - \left\langle f(\mathbf{x_i}; \hat{\boldsymbol{\theta}}_{\mathscr{D}}) \right\rangle_{\mathscr{D}} \right)^2 , \tag{1.55}$$

and *variance*:

$$\text{var} = \sum_{i=1}^{N} \left\langle \left(f(\mathbf{x_i}; \hat{\boldsymbol{\theta}}_{\mathscr{D}}) - \left\langle f(\mathbf{x_i}; \hat{\boldsymbol{\theta}}_{\mathscr{D}}) \right\rangle_{\mathscr{D}} \right)^2 \right\rangle_{\mathscr{D}} . \tag{1.56}$$

Bias represents how well our model would perform if we had an infinite amount of data, and the variance tells us how much our model fluctuates because of the finite amount of samples we have. Increasing the number of parameters will reduce the bias, but at some moment, the variance will increase. This is the central concept in ML, the trade-off between the complexity of a model and the amount of data we have to train it on. Because, in practice, we are usually presented with a limited amount of data. Choosing a less complex model with a higher bias will often lead to less variance and better generalization on new data points.

Now that we have covered the basic intuition behind ML and DL, we will specify one simple model we will use later in the thesis, *logistic regression*. Let us consider a case where our target variable y can take a value $m = 0, \ldots, M-1$ from one of the M classes. We want to define a function that, given an input, returns a probability that it belongs to one of the M classes. One such function is the *sigmoid* function:

$$\sigma(s) = \frac{1}{1 + \exp(-s)} \tag{1.57}$$

Let us start first with a simple case of two classes $y_i = \{0, 1\}$. Then a probability that given data point $\mathbf{x}_i$ belongs to a category is:

$$p(y_i = 1 | \mathbf{x}_i, \boldsymbol{\theta}) = \frac{1}{1 + \exp\left(-\mathbf{x}_i^T \boldsymbol{\theta}\right)} \tag{1.58}$$

$$p(y_i = 0 | \mathbf{x}_i, \boldsymbol{\theta}) = 1 - p(y_i = 1 | \mathbf{x}_i, \boldsymbol{\theta}) \tag{1.59}$$

The most appropriate loss function would be cross-entropy:

$$\mathscr{L}_{\text{BCE}} = -\sum_{i=1}^{N} y_i \log\left(\sigma(\mathbf{x}_i^T \boldsymbol{\theta})\right) + (1 - y_i) \log\left(1 - \sigma(\mathbf{x}_i^T \boldsymbol{\theta})\right) \tag{1.60}$$

We can minimize this loss function, but there is no simple closed-form solution and some of the minimization methods we have discussed need to be applied. Moving to a general case when instead of 2, we have M classes, and following one-hot encoding, we can write individual probabilities for each class as:

$$p(y_{i,m'} = 1|\mathbf{x}_i, \{\boldsymbol{\theta}\}_{k=0}^{M-1}) = \frac{\exp\left(-\mathbf{x}_i^T \boldsymbol{\theta}_{m'}\right)}{\sum_{m=0}^{M-1} \exp\left(-\mathbf{x}_i^T \boldsymbol{\theta}_m\right)} \tag{1.61}$$

This is known as *SoftMax* function, and the appropriate cross-entropy is:

$$\begin{aligned}\mathscr{L}_{\text{CCE}} = -\sum_{i=1}^{N}\sum_{m=0}^{M-1} y_{i,m} \log\big(p(y_{i,m} = 1|\mathbf{x}_i, \boldsymbol{\theta}_m)\big) \\ + (1 - y_{i,m}) \log\big(1 - (p(y_{i,m} = 1|\mathbf{x}_i, \boldsymbol{\theta}_m))\big)\end{aligned} \tag{1.62}$$

1.3.1. Neural networks for ML

Models that we have considered so far are simple ML models that form a basis for any further improvements. The introduction of neural networks catapulted the field of ML and DL to the heights of today.

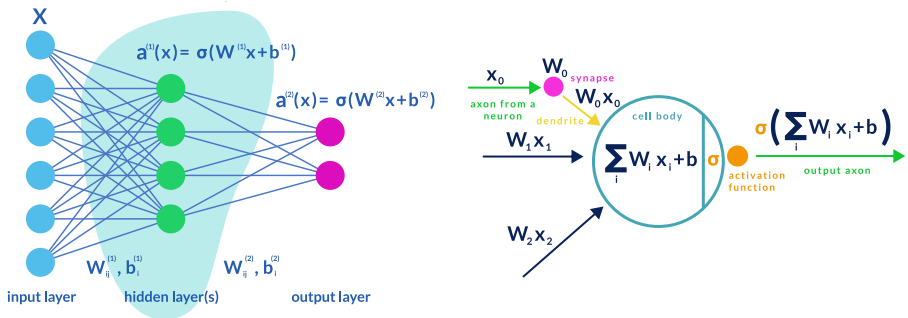


Figure 1.9: On the left is an illustration of a fully connected neural network with a single hidden layer and two layers in total. On the right is a neuron, an elementary building block of neural networks.

The fundamental building element of a neural net is a neuron i, Fig.(1.9) that takes as an input d dimensional vector of features $\mathbf{x} = (x_1, x_2, \ldots, x_d)$ and produces a scalar output $a_i(\mathbf{x})$. A neural network is formed by stacking many neurons on top of each other to form a single layer. Then multiple layers are stitched together to form the whole network. Stacking multiple layers enhances our neural networks' expressivity, which helps us approximate complex functions. The existence of a universal approximation theorem states that a neural network with a single layer can approximate

any "nice" and continuous function with arbitrary accuracy. The first layer is called an input layer, the last layer is called an output layer, and any layer(s) in-between is(are) called hidden layer(s). What makes neural networks work is an introduction of some non-linearity between layers, *activation function*. We usually take the same activation function for all the neurons in a single layer. Going from one to the other layer, first, the linear transformation of the form is computed:

$$z^i = \boldsymbol{\omega}^i \cdot \mathbf{x} + \mathbf{b}^i = \mathbf{x}^T \cdot \mathbf{w}^i \,, \tag{1.63}$$

where weight vector $\boldsymbol{\omega}^i = (\omega_1^1, \omega_2^1, \ldots, \omega_d^1)$, $\mathbf{x} = (1, \boldsymbol{x})$ and $\mathbf{w}^i = (b^i, \boldsymbol{\omega}^i)$. After the linear transformation, the activation function is applied to get a post-activation value of a layer:

$$a_i(\mathbf{x}) = \sigma_i(z^i) \tag{1.64}$$

There are many possibilities for activation functions; some of the popular ones are step-function, sigmoid, hyperbolic tangent, rectified linear units (ReLUs), leaky rectified linear units (leaky ReLUs), and exponential linear units (ELUs)[5]. What we choose as an activation function will impact the performance of our network and greatly depend on a specific task we are trying to solve, which we will later see in Chapter 5. Another consideration should be taking a derivative of activation functions as they are required for any gradient-based function minimization methods.

If we remember from our previous sections how to train a network, we will need to find a derivative of a loss function with respect to all the weights for any gradient-based method. At first, this seems like a daunting task. However, luckily, there is a very nice and elegant algorithm that can help us find all of the desired derivatives easily, *backpropagation*. Behind the fancy name is nothing more complicated than a simple chain rule for differentiation. Because this is crucial background knowledge to the chapter, we discuss it here. In order to fully understand this backbone of an algorithm, let us set the stage first. We will take a network with L layers labeled with $l = 1, \ldots, L$. Weights $\omega_{j,k}^l$ connect k-th neuron from $l-1$ layer to a j-th neuron in layer l. The bias associated with this neuron is b_j^l. Then we can write a post-activation value a_j^l of this neuron as:

$$a_j^l = \sigma\left(\sum_k \omega_{j,k}^l a_k^{l-1} + b_j^l\right) = \sigma(z_j^l) \tag{1.65}$$

where the linear combination part, the pre-activation value, is defined as:

$$z_j^l = \sum_k \omega_{j,k}^l a_k^{l-1} + b_j^l \tag{1.66}$$

Let us think about how we compute the loss function. Directly the value of a loss function depends on the post-activation values a_j^L from L-th layer, but these values indirectly depend on the post-activations from the previous layers. This is the crucial observation to set up a backpropagation algorithm. Define the error Δ_j^L of j-th

neuron in the final L-th layer as a partial derivative of the loss function with respect to the weighted input z_j^L:

$$\Delta_j^L = \frac{\partial \mathscr{L}}{\partial z_j^L} \tag{1.67}$$

Analogous to this, we can define the error for any neuron j in arbitrary layer l as:

$$\Delta_j^l = \frac{\partial \mathscr{L}}{\partial z_j^l} = \frac{\partial \mathscr{L}}{\partial a_j^l}\sigma'(z_j^l) \tag{1.68}$$

we can also make the chain rule different in order to get the following:

$$\Delta_j^l = \frac{\partial \mathscr{L}}{\partial z_j^l} = \frac{\partial \mathscr{L}}{\partial b_j^l}\frac{\partial b_j^l}{\partial z_j^l} = \frac{\partial \mathscr{L}}{\partial b_j^l} \tag{1.69}$$

where we have used Eq.(1.66) to find $\frac{\partial b_j^l}{\partial z_j^l} = 1$. As we stated before, the layered structure of the network ensures that the layer error $l+1$ depends on the post activations from layer l, and we can use the chain rule to expand:

$$\begin{aligned}\Delta_j^l = \frac{\partial \mathscr{L}}{\partial z_j^l} &= \sum_k \frac{\partial \mathscr{L}}{\partial z_k^{l+1}}\frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \sum_k \Delta_k^{l+1}\frac{\partial z_k^{l+1}}{\partial z_j^l} \\ &= \left(\sum_k \Delta_k^{l+1}\omega_{k,j}^{l+1}\right)\sigma'(z_j^l)\end{aligned} \tag{1.70}$$

The final equation is:

$$\frac{\partial \mathscr{L}}{\partial \omega_{j,k}^l} = \frac{\partial \mathscr{L}}{\partial z_j^k}\frac{\partial z_j^k}{\partial \omega_{j,k}^l} = \Delta_j^l a_k^{l-1} \tag{1.71}$$

Now we have all the necessary ingredients to state the entire backpropagation algorithm:

Algorithm 2 The backpropagation algorithm

1: Calculate the post-activation values of all neurons in the input layer a_j^1
2: Now using Eq.(1.65) compute all the z^l and a^l values until the last layer
3: Calculate the error of the final layer using Eq.(1.68); for this, we will need to compute the analytical form of loss and activation functions manually
4: Using Eq.(1.70) we can propagate error backwards in order to calculate Δ_j^l
5: The final step is using Eqs.(1.69, 1.71) in order to calculate the desired derivatives $\frac{\partial \mathscr{L}}{\partial w_{j,k}^l}$ and $\frac{\partial \mathscr{L}}{\partial b_j^l}$

The name backpropagation now makes sense; we are using a single forward pass through the network to compute linear combinations and post-activations, and then by backtracking through the network, we compute all of the derivatives. This specific nature of backpropagation makes it highly efficient when implemented on modern GPU units. The immediate relevance to the work in this thesis is the appearance of σ' in Eq.(5.2). A core part of chapter 5 is the exploration of the effects of saturation, domain regions where $\sigma' = 0$ and therefore negligible gradients, on the final training efficiency.

1.3.2. Neural quantum states

In order to motivate this next class of neural networks, we use some topics from quantum mechanics to guide us. Let us say that we have an isolated quantum system of spins 1/2 in a chain of length N. Then we can expand any wave function in some arbitrary basis that will have 2^N coefficients. For example, consider a $1D$ system with 40 spins. Directly writing down all the coefficients alone would be an infeasible task; there are $2^40 \sim 10^12$ different coefficients, and just writing them down would take up more than 40TB of space. If we want to study real-life systems with many more particles and in many more dimensions like $2D$ or $3D$, we need to find some other methods of doing it. This should not be the end; usually, only a small part of a Hilbert space is of relevance, and this fact can help us a lot. We can apply some of the *variational methods* that try to find the optimal representation of the quantum state, one that will encompass all the necessary features and be computationally friendly. We can write the basis expansion as:

$$\left|\Psi_{\boldsymbol{\theta}}\right\rangle = \sum_{s=1}^{2^N} \Psi_{\boldsymbol{\theta}}(\mathbf{s}) \left|s\right\rangle \tag{1.72}$$

where $\Psi_{\boldsymbol{\theta}}(\mathbf{s}) = \left\langle s\middle|\Psi_{\boldsymbol{\theta}}\right\rangle$ and our goal would be to find the best $\Psi_{\boldsymbol{\theta}}(\mathbf{s})$ that approximate a ground state but $\dim[\boldsymbol{\theta}] \ll \dim[s]$. We can then use this proposed model for the wave function to compute the expectation values of desired operators in polynomial time. To cast the usual complex wave function computations in terms of probabilities, we use the following local estimator method to compute the expectation value of some arbitrary operator $\hat{O}$ [5]:

$$\begin{aligned}\left\langle \hat{O} \right\rangle &= \frac{\left\langle \Psi_{\boldsymbol{\theta}}\middle|\hat{O}\middle|\Psi_{\boldsymbol{\theta}}\right\rangle}{\left\langle \Psi_{\boldsymbol{\theta}}\middle|\Psi_{\boldsymbol{\theta}}\right\rangle} \\ &= \frac{\sum_{s,s'} \left\langle \Psi_{\boldsymbol{\theta}}\middle|s\right\rangle \left\langle s\middle|\hat{O}\middle|s'\right\rangle \left\langle s'\middle|\Psi_{\boldsymbol{\theta}}\right\rangle}{\sum_s \left|\left\langle \Psi_{\boldsymbol{\theta}}\middle|s\right\rangle\right|^2} \\ &= \frac{\sum_s \left|\left\langle \Psi_{\boldsymbol{\theta}}\middle|s\right\rangle\right|^2 \sum_{'s} \left\langle s\middle|\hat{O}\middle|s'\right\rangle \frac{\left\langle s'\middle|\Psi_{\boldsymbol{\theta}}\right\rangle}{\left\langle s\middle|\Psi_{\boldsymbol{\theta}}\right\rangle}}{\sum_s \left|\left\langle \Psi_{\boldsymbol{\theta}}\middle|s\right\rangle\right|^2}\end{aligned} \tag{1.73}$$

We can identify two terms here:

$$p(\mathbf{s}) = \frac{|\langle \Psi_{\boldsymbol{\theta}} | s \rangle|^2}{\sum_s |\langle \Psi_{\boldsymbol{\theta}} | s \rangle|^2} \tag{1.74}$$

$$O_{\text{loc}}(s) = \sum_{s'} \langle s | \hat{O} | s' \rangle \frac{\langle s' | \Psi_{\boldsymbol{\theta}} \rangle}{\langle s | \Psi_{\boldsymbol{\theta}} \rangle} \tag{1.75}$$

The first is the usual quantum mechanics probability density. The second one $O_{\text{loc}}(\mathbf{s})$ is called a *local estimator* of $\hat{O}$. Therefore we can write a quantum mechanical expectation as a classical expectation value:

$$\left\langle \hat{O} \right\rangle = \sum_s p(\mathbf{s}) O_{\text{loc}}(\mathbf{s}) = \left\langle O_{\text{loc}}(\mathbf{s}) \right\rangle_p \tag{1.76}$$

The procedure of how to estimate the expectation value of any operators in the form of Eq.(1.76) is very reminiscent of Eq.(1.21) so we can write it as:

$$\left\langle \hat{O} \right\rangle \approx \frac{1}{N_{\text{samp}}} \sum_{i=1}^{N_{\text{samp}}} O_{\text{loc}}(\mathbf{s}^i) \tag{1.77}$$

Now we are back to the conventional statistical physics that we reviewed earlier. We now know how to compute this by constructing a Markov Chain and using Monte Carlo simulation to compute the expectation value.

What remains is to make a suitable variational ansatz $\Psi_{\boldsymbol{\theta}}(\mathbf{s})$. This is where we combine ML with Monte Carlo, an insight from Carleo and Troyer [4]. We will use the *restricted Boltzmann machines* (RBM) for this job. Neural network architecture is called the *neural quantum state* (NQS).

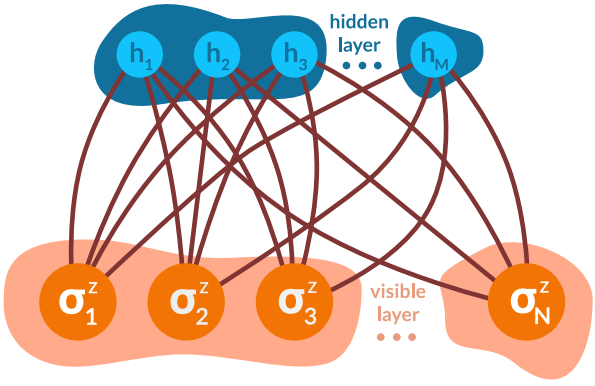


Figure 1.10: Illustration of a restricted Boltzmann machine (RBM) neural network. We use RBM as a variational ansatz to represent a wave function of N spins with $s = (\sigma_1^z, \sigma_2^z, \ldots, \sigma_N^z)$ and M hidden units $h = (h_1, h_2, \ldots, h_M)$. There is general proof that a sufficiently dense ($\alpha \gg 1$) RBM can approximate any probability distribution [6].

RBM networks are fully connected two-layer networks with one visible and one hidden layer. The visible layer has N spins, and the hidden layer has M spins. We will label the ratio of the number of hidden vs. visible neurons as $\alpha = \frac{M}{N}$. The insight of Carleo and Troyer was that the intrinsically non-local correlations of RBM could lead to a significantly more compact representation of many-body quantum states. Then the ansatz wave function is given by:

$$\Psi_{\boldsymbol{\theta}}(\mathbf{s}) = \sum_{h} \exp\left(\mathbf{b}_v^\dagger \cdot \mathbf{s} + \mathbf{b}_h^\dagger \cdot \mathbf{h} + \mathbf{h}^\dagger \mathbf{W} \mathbf{s}\right). \tag{1.78}$$

Because of the network architecture, we can trace out hidden variables and get the following:

$$\Psi_{\boldsymbol{\theta}}(\mathbf{s}) = \exp\left(\mathbf{b}_v^\dagger \cdot \mathbf{s}\right) \prod_{i=1}^{M} 2\cosh\left(\mathbf{b}_{h,i} + \mathbf{W}_i \cdot \mathbf{s}\right), \tag{1.79}$$

where $\mathbf{b}_{h,i}$ and $\mathbf{W}_i$ are i-th hidden bias and weight matrix row. Now we have a model wave function that will convert our spin configuration into a desired probability.

With this ansatz, we are back in familiar territory to find the ground state energy of some Hamiltonian:

$$E(\boldsymbol{\theta}) = \left\langle \Psi_{\boldsymbol{\theta}} \middle| \hat{H} \middle| \Psi_{\boldsymbol{\theta}} \right\rangle, \tag{1.80}$$

in terms of ML, this will be our loss function. We must also remember that this variationally obtained ground state energy is $E(\boldsymbol{\theta}) \geq E_0$, but this is strictly true when expectation values are calculated exactly.
We can exchange operator $\hat{O}$ for the energy in Eq.(1.77) in order to get stochastically approximated energy:

$$E(\boldsymbol{\theta}) \approx \frac{1}{N_{\text{samp}}} \sum_{i=1}^{N_{\text{samp}}} E_{\text{loc}}(\mathbf{s}^i) \tag{1.81}$$

where we have defined our local energy as $E_{\text{loc}}(\mathbf{s}) = \sum_{s'} \langle s|\hat{H}|s\rangle \frac{\langle s'|\Psi\rangle}{\langle s|\Psi\rangle}$. We will use gradient-based optimization methods to minimize our loss function. The energy gradient with respect to parameters is:

$$\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i} = 2\,\text{Re}\left[\left\langle E_{\text{loc}}(\mathbf{s}) O_i^\star(\mathbf{s})\right\rangle - \left\langle E_{\text{loc}}(\mathbf{s})\right\rangle \left\langle O_i^\star(\mathbf{s})\right\rangle\right] \tag{1.82}$$

where θ_i is the i-th parameter and operator $\hat{O}_i$ is defined as:

$$O_i(\mathbf{s}) = \frac{\partial}{\partial \theta_i} \log\left\langle s \middle| \Psi_{\boldsymbol{\theta}} \right\rangle = \langle s|\hat{O}_i|s\rangle \tag{1.83}$$

Now we have all the necessary ingredients to formulate the final algorithm for finding the ground state of a Hamiltonian $\hat{H}$ using RBM and variational method:

Compared to the current standard numerical methods, NQS can have the same or better accuracy[4]. Compared with MPS, PEPS, and DMRG for transverse-field Ising

Algorithm 3 Finding ground state with NQS

Randomly initialize all the parameters of a neural network $\boldsymbol{\theta}$
for $i = 1$ to N_{steps} **do**
  Generate N_{samp} samples using Markov chain
  Calculate the gradient of the energy $\frac{\partial E(\boldsymbol{\theta})}{\partial \theta_i}$ using Eq.(1.82)
  Update parameters using SGD or some other minimization technique
end for
return Optimized parameters $\hat{\boldsymbol{\theta}}$

(TFI) and anti-ferromagnetic Heisenberg models (AFH), NQS achieved better accuracy for sufficiently large α ration. AFH, when compared to DMRG, NQS with $\alpha = 4$ managed to outperform DMRG with the bond dimension of ~ 160, pointing towards a much more compact representation of a many-body wave function. This also has practical implications, as fewer variational parameters are easier to optimize. Even when applied at the critical point, NQS managed to get the accuracy of the state-of-the-art methods or even better, albeit with slowed down converged, which is expected near the criticality. The compact nature and ability to express the wave function near the critical point prompted us to use it to analyze entanglement entropy for lattice gauge fields explored in chapter 4 of the thesis.

1.4. This thesis

In the introduction, we have covered the basic ideas used later in this thesis. We started with introductory topics in thermodynamics and statistical physics, then moved to a basic introduction to Monte Carlo methods and all the required knowledge to understand our physical system's simulation design and analyze the results. The proceeding section was dedicated to the basics of machine learning, deep learning, and appropriate selection of model, loss function, and minimization method. The last section culminated in a synergy of the previously mentioned topics by combining quantum physics, Monte Carlo methods, and neural networks in neural quantum states that we used to find the ground state and its energy of lattice gauge theories.

1.4.1. Chapter 1 - Thermalization in quantum systems

The properties of closed unitary quantum systems and how they exactly thermalize have been one of the leading research questions in statistical physics for a long time. The puzzle is that a thermal ensemble is formally a mixed state, but a mixed state can never arise from unitary evolution from a pure state. The usual answer to how they thermalize is the eigenstate thermalization hypothesis ("ETH") [7]. The hypothesis is that in generic quantum theory with many degrees of freedom, most observables will have a particular form of matrix elements after averaging, and observable will *appear* to thermalize. However, recently [8] showed that ETH has to be taken with care. Even in free field theory, there are operators that *appear* to relax, called operator thermalization hypothesis (OTH). Given a particular no-go condition, the retarded Green's function will typically decay exponentially unless the condition is met. Finding an operator that will satisfy this condition in a general non-integrable theory is challenging but possible. On the other hand, this job is more straightforward in integrable theories due to the extensive number of conserved quantities. We work in the transverse field Ising (TFI) model where we compare a specially designed operator Γ that will satisfy the no-go condition with the Pauli σ^z operator that does not satisfy it. Through the examples, we show the differences and similarities of ETH and OTH and how, despite TFI being an integrable theory, σ^z will relax after the perturbation. Also, we have demonstrated how the no-go condition is a feature of integrability, and any minor deviation from integrability will cause Γ to relax. Our results were later confirmed in [9].

1.4.2. Chapter 2 - Symmetry restoration through "registry"

Starting from the simple Wegner gauge theory [10, 11], Fradkin and Shenker [12] discovered that when an added matter field is "in the fundamental", meaning that

there is an additional Higgs field that is also governed by Z_2 symmetry as the gauge fields, the Higgs phase and confining phase become one, without a phase transition. In this chapter, we propose a straightforward generalization of their lattice gauge theory that could serve as a candidate for a highly entangled state of matter. We will consider adding multiple Z_2 and $O(N)$ matter fields on the lattice and gauging them with a common Z_2 field. It will be shown how, in such a case, the Higgs phase becomes separate from the confining phase. It will be characterized by the "registry" order parameter, which turns out to be gauge invariant $p = 2^{N_{\text{rep}-1}}$ Potts type symmetry, where N_{rep} is the number of matter field copies. Interpretation for this type of symmetry is that different matter copies align their vectors locally in strictly parallel or anti-parallel ways, even in the case of continuous $O(N)$ matter fields. These theories will be studied using Monte-Carlo simulation on a $3D$ grid using the Metropolis-Hastings algorithm and annealing techniques to improve the convergence near the critical point. From the simulation results, we can discover some unidentified "pseudo-universality" associated with the form of the phase diagram for various numbers of matter field theories.

1.4.3. Chapter 3 - Entanglement entropy of lattice gauge theories

Building further on the work from the previous chapter, we will study entanglement entropy in the neural network representation of the above lattice gauge theories, now considered as quantum theories in one lower dimension. Following the seminal work of Carleo and Troeyer [4], we will construct neural quantum states as the representation of our theory using a variational wavefunction based on Restricted Boltzmann Machines used in Machine Learning. Using ideas from tensor networks that the bond dimension represents the upper bound on the amount of entanglement a state can have, we will postulate that by increasing the number of matter fields, ground state entanglement entropy of our lattice gauge theory will increase as the ratio of hidden vs. visible nodes. We have tested our hypothesis in the case of $2,3$, and 4 matter fields. Within the achievable computational limits, the results are puzzling. Even though increasing the number of variational parameters improved the energy of the ground state, the impact on the entanglement entropy is less than obvious. Curves of entanglement entropy for different system sizes look the same up to the statistical errors.

1

1.4.4. Chapter 4 - Phase space and efficient learning of deep neural networks

This chapter combines some statistical physics insights into machine learning with the computational mechanics of deep random feedforward neural networks. In recent times with the ever-growing amount of available data, neural networks have become one of the de-facto methods for analyzing and processing vast amounts of data [13]. One of the reasons why these methods became so popular is their ability to express any function with a relatively small number of parameters [5] and the ease with which this *expressivity* can be increased by adding more depth. This easy fix does not come for free; deep neural networks generally require more training computations. Specifically, they suffer from exploding or vanishing derivatives in optimizing the parameters. The phase space of deep random feedforward neural networks is characterized by the variance of initial weights and the variance of initial bias. Following previous work [14, 15] that demonstrated the existence of order-to-chaos regime change in this phase space, we will examine the behavior of the pre- and post-activations in terms of their distributions and also final accuracy on classification task such are MNIST and CIFAR10. The phase boundary dividing these two regimes is called the edge of chaos (EOC). We demonstrate that for the tanh activation function, not all points along the EOC yield the same learning efficiency. In the case of shallow and narrow neural networks, we define the line of uniformity (LOU), a set of points for which the final layer post-activation values are distributed uniformly, i.e., with maximal entropy. We show that moving away to the right from LOU and drastically increasing initial variances means that gradient saturation will start impeding optimization over parameters, i.e., the learning process.