Machine-Learning-Based Mapping Of The Ising Model To Rough Surfaces

Author: Sebastian Korsak

Registration Number: 09319012

Supervisors: Pe. Dr. F. Diakonos, Pe. Dr. V. Constantoudis

Master Thesis

National Technical University of Athens 2021

Faculty of Applied Mathematics and Physics

Master of Mathematical Modeling in Modern Technologies and Finance

An author's note

This thesis was written in the dark year of 2021, the year where a pandemic ruined humanity and a lot of people had to do very essential and unpleasant changes in their lives. And probably these changes were more important than the virus itself. Therefore, the first thing I want to say is that I wish from the bottom of my heart all the other students in the near future will back in their normal lives again and write beautiful scientific essays and publications in a more healthy environment than the reality we live now. And apart from scientific activity, I wish them to have this beauty in every aspect of their lives.

For this thesis, I want to thank my professors Pf. dr. F. Diakonos and Pf. dr. V. Constantoudis, for their collaboration and support. Also, I want to thank the whole Technical University and National Kapodistrian University of Athens that gave me all the knowledge I needed to write down the ideas of this thesis. It is notable that all the people in University made a very big effort to keep the knowledge alive, even if the conditions were adverse.

~ Sebastian Korsak, July 2021

Abstract

When we started this thesis, our initial idea was to investigate the interaction of two scientific fields: Machine Learning and Complex Systems. This interaction has two directions. The first one is to investigate how Machine Learning methods can be applied in Complex Systems. A common example of this interaction is about determining phase transitions in systems that it is difficult to find an order parameter, or even if we can define a kind of this parameter, it is still very difficult to understand analytically the phase transition. On the other hand, Deep Learning offers a great variety of mathematical tools with applications in almost every scientific field, which, however, luck of a strict theoretical understanding. Therefore, ideas borrowed from Complex Systems can be helpful to lead us in a deeper understanding.

In this thesis, we investigate Ising model, which is maybe one of the most popular Complex Systems. Our main goal is to create an alternative algorithm to Metropolis-Hastings, which is able to produce Ising lattices for different temperatures using Rough Surfaces simulation. The main advantage of this simulation is its speed, and the fact that it is able to produce uncorrelated lattices. Moreover, it offers a new geometrical perspective to Ising model, since we don't need to define a kind of Hamiltonian (or energy) function to use Rough Surfaces simulation. Gaussian surfaces simulation has only geometrical parameters as input (i.e. mean, variance, correlation length, threshold) and thus we can investigate and reveal new geometrical aspects of Ising model.

To reach our goal, we need to have some diagnostics to reassure that our algorithm works correctly. Consequently, we consider some complexity measures that can give us some geometrical interpretation of the aggregates which appear in the critical temperature. Having these complexity measures in combination to other statistical quantities like magnetization or susceptibility, we can use them as diagnostics for our algorithm. It is important to have as many measures as possible to reassure that our algorithm is valid.

The role of Machine Learning in this thesis is to use it as an interpolation technique to find a correspondence between Metropolis-Hastings algorithm and Rough Surfaces simulation. Particularly, we start by comparing lattices from each simulation in respect to two similarity measures, and then we use Machine Learning to learn our data. This is something that can be done quite well with classical regression techniques. However, Machine Learning algorithms offer us a better accuracy, so it makes sense to compare classical regression and Machine Learning models to see which of them can lead us to better results. Additionally, some of the complexity measures are inspired from Boltzmann Machine algorithms. Although we don't present a systematic study of Boltzmann Machines, their theoretical understanding was crucial to define these measures.

Contents

List of Figures

Prologue

In this thesis we work with Ising model, which is a model of a magnet and one of the most popular Complex Systems. A very good introductory book about Ising model is this of Barry M. McCoy and Tai Tsun Wu with title *"The Two-Dimensional Ising Model"* [3], where the authors do a very deep theoretical introduction to this model. Another very good book related to computational methods of Ising model is this of M. E. J. Newman and G. T. Barkema *"Monte Carlo Methods in Statistical Physics"* [1]. This is a very good reference to understand every aspect of Metropolis-Hastings algorithm, and we based a big part of this thesis to it. Nevertheless, we can see references of Ising model in almost every book of Statistical Mechanics or Complex Systems. For instance, the book of Sacha Friedli and Yvan Velenik *"Statistical Mechanics of Lattice Systems"* [2], which does a very good introduction to discrete lattice systems, or the book of Chjan Lim and Joseph Nebus *"Vorticity, Statistical Mechanics, and Monte Carlo Simulation"* [6] which also focuses on computational methods. Another useful reference is this of Peter Eastman from Stanford University [5], which is a website made for educational reasons and offers a simplified introduction to the complex ideas of Statistical Mechanics for people who are new to them.

About Machine Learning (or Pattern Recognition), it is a scientific field which is well spread in our society. Applications of Machine Learning can be found in our daily lives (i.e. applications in mobile phones, cameras, social media) or even in scientific organization like NASA or CERN (*"Machine learning and the physical sciences"*, Giuseppe Carleo et al. RMP2019 [20]). Surprisingly, there are applications in theoretical physics also. A very good essay on the theoretical applications (and not only) in physical sciences is the paper of Yang Tang et al. *"Introduction to Focus Issue: When machine learning meets complex systems: Networks, chaos, and nonlinear dynamics"* [4], in which the authors discuss all the research of this interaction between these two scientific fields that exists until now. From smartphones to theoretical physics, there is one thing in common: mathematical methods. Therefore, Machine Learning can be seen as a collection of computational mathematical methods that help machines to act like they have some kind of intelligence. In the book of Hertz and Krogh "*Introduction to the Theory of Neural Computation*" [22], there is a chapter about the history of what we call Neural Networks, which started from energy based models and shared some common ideas from Complex Systems and Physics generally. The starting of modeling real biological neural networks that exist in our brain was the start point of artificial neural networks, and the process of "copying" nature to build new mathematical models is called "*biomementism*". Other classical books for people who are new in Pattern Recognition are these of Bishop [8], and Duda and Hart [9].

One chapter of this thesis is related to the definition of Complexity. Particularly, we tried to define and compute some measures of Complexity in Ising model's configurations for different values of temperature. It is notable that Complexity is an intuition-based term, and thus there are a lot of different ways to define it. In [18]

James Ladyman and Karoline Wiesner do a discussion about "*What is Complexity?*". It looks like that even if there is a lot of research on Complex Systems, there is still the question about what really is "complex". In this essay, authors write that the main factors of complexity are: numerosity, disorder and diversity, feedback, non-equilibrium, spontaneous order and self-organization, non-linearity, robustness, nested structure and modularity, history and memory or adaptive behavior. Additionally, there is another paper of R. Wackerbauer et al. "*A Comparative Classification of Complexity Measures*" [11], in which the authors categorize complexity measures as structural (or dynamical) based on homogenous (or generating) partitions. In this thesis with had as start point the paper of Ana Lorena et al. "*How Complex Is Your Classification Problem?: A Survey on Measuring Classification Complexity*" [10], where the authors define some complexity measures for classification problems. In this manner, they see complexity in terms of data science and the main question they try to solve is "how complex are the data we feed a classifier?" or "how easily we can separate classes?". All these measures offer a new perspective in the definition of complexity, since the way they face the problem is different from the formal definition of algorithmic complexity by Kolmogorov in 1965. The algorithmic (Kolmogorov-Chaitin) complexity of a pattern [12], [13] is essentially given by the length of the shortest algorithm capable of reproducing the pattern. A more recent work that combines complexity measures and Ising model is this of Andrey A. Bagrov et al. "*Multiscale structural complexity of natural patterns*" [14], in which the authors attempt a quantitative definition of structural complexity of patterns in terms of Renormalization Group flow.

The main part of this thesis is about reconstructing Ising model with Gaussian Rough Surfaces simulation. This may sounds a bit strange, because Ising model is a dynamical model of a magnet, whereas Gaussian Surfaces simulation refers to static surface morphologies. However, even if these two phenomena look so different, we can find a relation between them. Barabasi and Stanley have written a book named "*Fractal Concepts in Surface Growth*" [16], which is a classical one for the simulation of Rough Surfaces and focuses on fractal methods. However, fractals are not the only procedure as there are works like this of Kim, Machiraju and Thompson "*Modeling Rough Surfaces*" [17], where they use discrete surface growth models which is a diffusion based approach. In this thesis we focus in Fourier Transform based methods and our main source and method is this which was proposed by Yang, Li, Wang, and Hong in their paper "*Numerical Simulation of 3D Rough Surfaces and Analysis of Interfacial Contact Characteristics*" [15], where they simulate both Gaussian and non-Gaussian surfaces. For the purpose of this thesis Gaussian Surfaces simulation is good enough to provide us results similar to Ising model.

This thesis is separated in three parts. In the first part, which is called "*Theoretical Prerequisites*", we do a small introduction to the theoretical background that somebody needs to understand the main concepts of this work. Consequently, in Chapter 1, we discuss about what is Ising model, what are phase transitions, we also do an introduction to Stochastic Processes so as to be able to understand Metropolis-Hastings algorithm. In Chapter 2, we do a small introduction to Machine Learning. Our main aim is to understand some basic concepts through some easy examples. Additionally, we examine the connection between Complex Systems and Machine Learning. In Chapter 3, there is an introduction to Rough Surfaces, the parameters that characterize them and the simulation that we use. Furthermore, we do a comparison between the lattices that can be produced from this simulation and Metropolis-Hasting's one. In the second part, which is named "*Main Part of Master's Thesis*" there are author's

work and results. Therefore, in Chapter 4 we computed the complexity measures that were proposed in [10], and we defined some new ones that are inspired from the algorithm of Boltzmann Machines. These complexity measures can be used as diagnostics for an Ising model simulation, or can either be computed in a Boltzmann Machine. In Chapter 5 we propose an algorithm that can simulate Ising model via Gaussian Rough Surfaces Simulation, and we discuss our results and the advantages of the algorithm. In the third part of this thesis, we have an appendix with the algorithms written in python.

Part I

Theoretical Prerequisites

What we call the beginning is often the end.
And to make an end is to make a beginning.
The end is where we start from.

- T.S. Eliot

1 Ising Model

The main subject of this thesis is Ising model. Therefore, it is essential to define this model and understand some basic concepts about it. Having done this, we will see how we can simulate its probability distribution and do computational work using Monte Carlo methods. The most popular Monte Carlo method that help us simulate from Boltzmann probability distribution is Metropolis-Hastings algorithm, which uses some elements from the theory of Stochastic Processes. In this chapter we will do a very fundamental introduction to all the mathematical tools we need.

1.1 What is Ising Model?

The Ising model is the model of a magnet. The topology of this model is the topology of a lattice with a magnetic dipole or spin to each site. In physics, we want simplicity, thus we assume that there are only two possible states for each spin $s_i = \pm 1$, representing up-pointing or down-pointing dipoles of unit magnitude. Another important simplification is that we assume that each spin can interact only with its nearest neighbors, and this interaction can be represented from a term of the product $s_i s_j$ in our Hamiltonian, and the strength of this interaction is symbolized by J. Additionally, we assume the existence of a magnetic filed B in our model. Therefore, we can write the following Hamiltonian,

$$H = -J \sum_{<ij>} s_i s_j - B \sum_i s_i \tag{1.1}$$

where the notation $< ij >$ indicates that the sites i and j appearing in the sum are nearest neighbors. In this thesis we do the assumption that $J = 1$ and $B = 0$. This is one of simplest models in Complex Systems. However, we fully understand its physics only in one dimension. Even if there is a lot of research on two or three dimensions, this model is complex enough that there is still a mystery about it. A very good book for lattice systems its this of S. Friedli and Y. Velenik [2], and for readers with a stronger mathematical background there is the book of Barry M. McCoy and Tai Tsun Wu [3]. However, in this introduction we have been influenced more from

the book of M. E. J. Newman and G. T. Barkema [1], which is related to Monte Carlo methods. Additionally, there is a very good introduction in Statistical Mechanics and Ising models at the site of Peter Eastman [5].

Since we do statistical mechanics, we need to define a probability distribution for our system. In the case of Ising model, the partition function is

$$Z = \sum_{s_1=\pm 1} \sum_{s_2=\pm 1} \cdots \sum_{s_N=\pm 1} \exp\left(\beta J \sum_{<ij>} s_i s_j + \beta B \sum_i s_i\right) \tag{1.2}$$

or in short

$$Z = \sum_{\{s_i\}} e^{-\beta H} \tag{1.3}$$

where $\beta = 1/kT$. Note that $k = 1.38 \times 10^{-23} J \cdot K^{-1}$ is the Boltzmann's constant and T is the temperature of our system.

The most convenient way to imagine this model of magnets, is by assuming that all the spins are located in a square lattice Fig. 1.1. The reason why this visual representation is helpful is because we can imagine them as binary images in our computer to have a better idea. Since each spin can take two values, there are a total of 2^N states for a lattice of N spins on it. Nevertheless, the set of all possible states of the Ising model, is just a subset of the set of all the possible binary images. Another

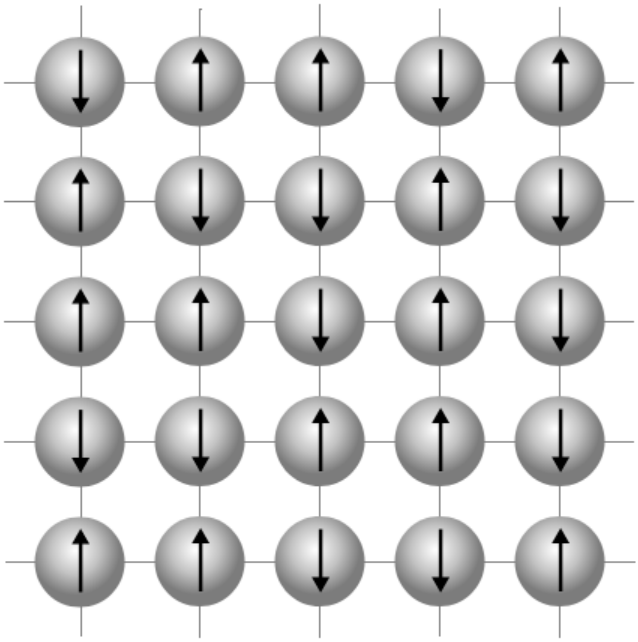


Figure 1.1: A visual representation of an Ising lattice [5].

reason why Ising model is so popular, is because it is a paradigm of phase transitions in physics. Perhaps, the most popular example of phase transitions is the behavior of water under the change of temperature. It is well known that water can exist in three different phases: solid, water or liquid. In the solid phase, the molecules form a crystal, but if we increase the temperature enough, there is a critical temperature (for about 0 Celsius) where the molecules no longer form any repeating pattern, and the ice becomes a liquid. If we increase the temperature more, there is a second critical point ($\sim$ 100 Celsius), where the water transforms from liquid to gas and the

molecules are no longer packed together. The most important characteristic of phase transitions is that these changes of our system happen very abruptly for some critical values of a specific order parameter (like temperature in case of water). Additionally, if we think in terms of temperature *and pressure*, there is another more interesting behavior in the triple point of water, where we have both solid, liquid and gas state at the same time.

In case of Ising model, to understand phase transitions we need to define the mean magnetization of our system $\langle M \rangle$, which is just the the average over the states

$$\langle M \rangle = \left\langle \sum_i s_i \right\rangle \tag{1.4}$$

or the mean magnetization per spin $\langle m \rangle$, which is just

$$\langle m \rangle = \frac{1}{N} \left\langle \sum_i s_i \right\rangle . \tag{1.5}$$

So for the Ising model, the phase transition also occurs under the change of temperature. As we can see in Fig. 1.2, for high temperatures $T > T_c$, our system has zero magnetization, which means that we have a symmetric configuration in our lattice. Nevertheless, for $T = T_c$ we have what is called 'spontaneous symmetry breaking', which means that in this critical point our configuration becomes fractal and there is no symmetry anymore. Specifically, we will see in the next chapters of this thesis that when our system is in its critical point, the configuration becomes the most complex in terms of its geometrical structure. Finally, for $T < T_c$ we have two possible values of magnetization with the same absolute value and different sign. For these temperatures the symmetry is broken. Note that the critical temperature for the Ising model is,

$$T_c = \frac{2J}{k \ln(1 + \sqrt{2})} \cong 2.268 \frac{J}{k}. \tag{1.6}$$

and in our simulation we will assume that $J = k = 1$.

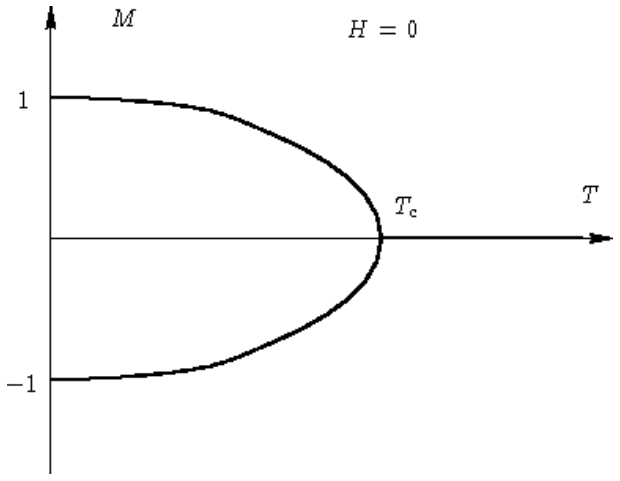


Figure 1.2: The phase transition of the Ising model.

Apart from magnetization, there are other statistical quantities of interest that can detect the phase transition of Ising model. These statistical quantities are derived from

the fluctuations of magnetization or internal energy by calculating the derivatives of the partition function. In this manner, we define the *magnetic susceptibility* as,

$$\frac{\partial \langle M \rangle}{\partial B} = \beta(\langle M^2 \rangle - \langle M \rangle^2) \tag{1.7}$$

or the *magnetic susceptibility per spin* as,

$$\chi = \frac{\beta}{N}(\langle M^2 \rangle - \langle M \rangle^2) = \beta N(\langle m^2 \rangle - \langle m \rangle^2) \tag{1.8}$$

Similarly we can calculate the specific heat per spin c from the energy fluctuations,

$$c = \frac{k\beta^2}{N}(\langle E^2 \rangle - \langle E \rangle^2) \tag{1.9}$$

All these definitions are important because they can help us to understand the phase transitions of Ising model, and also their dependence on temperature is a good indicator to show if we run the Monte Carlo simulation correctly.

At this point it is clear that phase transitions play an important role in Statistical Mechanics. However, there are different kinds of phase transitions that can occur in a system of interest. To understand the classification of phase transitions, we need to start from the definition of intensive and extensive variables [6]:

- An **intensive variable** is a property inherent to the pieces of the model physically these are properties like the conductivity or the density of a substance, or the inverse temperature at which phase transitions occur.
- An **extensive variable** depends on the amount of material present the total mass, or charge, or in our model the net vorticity of a fluid.

Given a set of n extensive variables $Q_1, Q_2, Q_3, \ldots, Q_n$, we can define $n-1$ densities as,

$$\rho_j = \frac{Q_j}{Q_n}, \text{ for } j = 1, \ldots, n-1 \tag{1.10}$$

(with, often, Q_n chosen to be a quantity like the volume or the number of particles in the system, making each ρ_i explicitly a density). Thus we can classify the phase transitions as

- **A phase transition of the first kind** is a discontinuity in one or more of these densities.
- **A second-order phase transition** is one in which each ρ_i is continuous, but the first derivative (and higher order derivatives) have a finite discontinuity.
- **A third-order phase transition** similarly is continuous and has a continuous first derivative, but the second and higher-order derivatives are discontinuous.

1.2 Mean Field Approximation

To have a clear analytical picture of Ising model we should be able to calculate the minimum energy of our system for every value of the energy H and temperature T. However, this is not always easy. Perhaps there are analytical solutions for the

one dimensional Ising model, but in two dimensions the situation is not that easy and thereby we must find a way to simplify our problem in order to end up in a solution that can give as an approximate result.

To start working with approximations, we write the following expression,

$$\begin{aligned} H &= -\sum_{i=0}^{N} s_i \left(B + J \sum_{<j>} s_j \right) \\ &= -\sum_{i=0}^{N} s_i \left(B + 2dJ \left\langle s_j \right\rangle' \right) \end{aligned} \tag{1.11}$$

where $\langle \dots \rangle'$ indicates that we are averaging over the $2d$ nearest neighbors of spin i. Mean field approximation tell us that we can replace $\langle s_j \rangle'$ with $\langle s_i \rangle$. If we do that, we can write,

$$H \approx -(B + 2dJM) \sum_{i=0}^{N} s_i \tag{1.12}$$

When we do this approximation, we change the topology of our lattice, and instead of a lattice as this in Fig. 1.1 where every spin interacts with its neighbors, now we have a fully connected network like this in Fig. 1.3 where all spins interact with all the other ones.

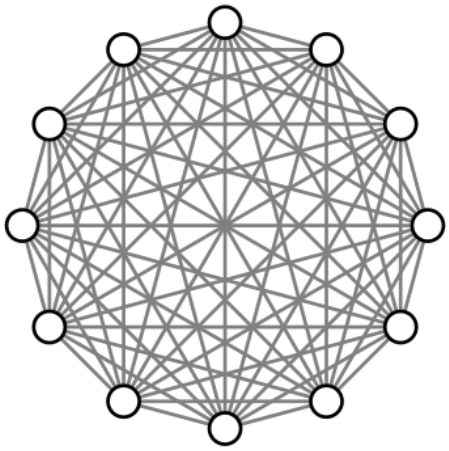


Figure 1.3: A mean field Ising configuration looks like a fully connected network. ([2])

Now let's come back in the general case of Ising model for a bit. Let's consider a single isolated spin, which has two possible states $s = 1$ and $s = -1$. Its partition function is,

$$\begin{aligned} \langle s \rangle &= (1)p(s = 1) + (-1)p(s = -1) \\ &= \frac{e^{B/kT} - e^{-B/kT}}{e^{B/kT} + e^{-B/kT}} \\ &= \tanh(B/kT) \end{aligned} \tag{1.13}$$

In case of mean field approximation we can substitute this external magnetic field B with an 'effective mean field' $B + 2dJM$, and thus,

$$M = \tanh\left(\frac{B + 2dJM}{kT} \right). \tag{1.14}$$

The magnetization appears in both parts of this equation, and thus to find the solutions of this system we need to solve the equation. The best way to that is by drawing the two curves $y = M$ against $y = \tanh\left(\frac{B+2dJM}{kT}\right)$ as we do in Fig. 1.4. For simplicity, we will only consider the case $J > 0$. According to [5], here are a few main possibilities:

- If $2dJ/kT \leq 1$, there is exactly one solution. It corresponds to $M = 0$ if $B = 0$ (a). Otherwise, M has the same sign as B (b). The system is magnetized by the applied field.

- If $2dJ/kT > 1$, there may be up to three solutions (c). Whichever one has the lowest free energy will be the stable one. As long as the temperature is low enough for energy to dominate over entropy, that will always be the one in which the system is most strongly magnetized in the direction of B. If $B = 0$, both the magnetized solutions have equal energy and are equally stable. (The solution near $M = 0$ is still unstable, being a state of high energy.)

- If B is sufficiently large there is only one solution (d). The only possibility is that the system is magnetized by the applied field.

For this thesis we only need the intuition of Mean Field Approximation to define some complexity measures that we will see in Chapter 2.

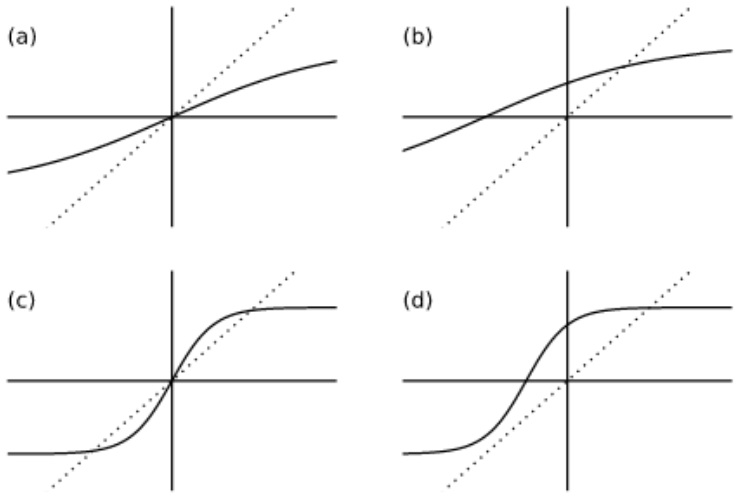


Figure 1.4: In this diagram we plot $y = M$ against $y = \tanh\left(\frac{B+2dJM}{kT}\right)$ for (a) $B = 0$, $\frac{2dJ}{kT} = \frac{1}{2}$, (b) $B = \frac{kT}{2}$, $\frac{2dJ}{kT} = \frac{1}{2}$, (c) $B = 0$, $\frac{2dJ}{kT} = 2$, (d) $B = kT$, $\frac{2dJ}{kT} = 2$. ([5])

1.3 Phase Transitions in Nonlinear Dynamics

In the theory of Dynamical Systems, phase transitions are called bifurcations and are related to structural stability of a system. Therefore, assuming that we have a system of nonlinear differential equations[1],

$$\dot{\boldsymbol{x}} = \boldsymbol{f}(\boldsymbol{x}) \tag{1.15}$$

[1] With dot $\cdot$ here we symbolize the time derivative.

with $\boldsymbol{f} \in C^1(E)$ where E is an open subset of $\mathbb{R}^n$, we say that the system is *structurally stable*, if the qualitative behavior remains the same for all nearby vector fields. In case that our vector field $\boldsymbol{f}$ is not structurally stable, it belongs to the bifurcation set in $C^1(E)$. A very good introductory book for this subject is this of Strogatz [30], and a more theoretical one is the book of Perko [29].

In this section we present two simple paradigms from the book of Strogatz [30]. The first one is the Saddle-Node Bifurcation, which is given from the first order system,

$$\dot{x} = r + x^2 \tag{1.16}$$

where r is a parameter, which may be positive, negative or zero. If we try to find the fixed points of this system, we will see that the equation $r + x^2 = 0$ has two solutions if $r < 0$, one solution if $r = 0$ and no solutions if $r > 0$. This creation or destruction of fixed points is what is called bifurcation of a system.

The bifurcation diagram is a plot of the fixed points versus the parameter r. In this case, we can see our bifurcation diagram in Fig 1.5, where with color blue we have drew the fixed points. It is clear that for $r < 0$, we have a stable and an unstable fixed point.

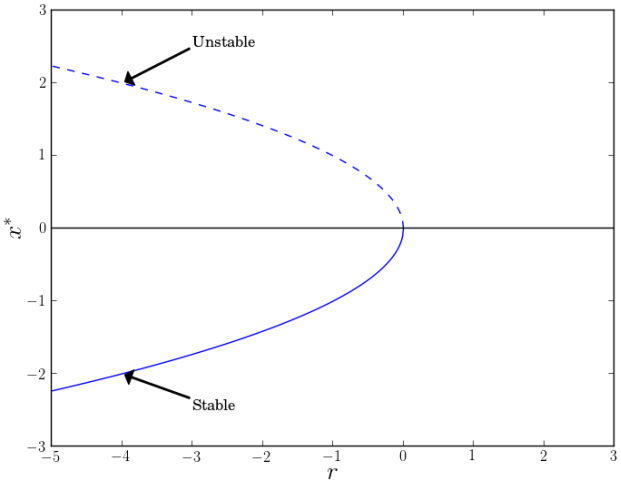


Figure 1.5: The Saddle-Node bifurcation diagram.

The second example is this of supercritical pitchfork bifurcation. And this is an important example because it reminds us Ising model. The normal form of the supercritical pitchfork bifurcation is,

$$\dot{x} = ax - x^3. \tag{1.17}$$

Note that this equation is invariant under the change $x \to -x$, which shows us a symmetry of our system. For $a \leq 0$ the origin is the only stable fixed point, whereas for $a > 0$ the origin becomes unstable and two new stable points appear on either side of the origin, symmetrically located at $x^* = \pm\sqrt{a}$.

In Fig. 1.6 we can see the Pitchfork bifurcation diagram. If we forget the mathematical symbolism, and we imagine x as the magnetization and a as the temperature, this diagram looks pretty similar to what we saw in Ising model (Fig. 1.2). The only

difference between Pitchfork bifurcation and the phase transition of the Ising model, is that they are derived with a completely different way. For the former we need a differential equation, whereas the later is derived by a lattice with spin interactions.

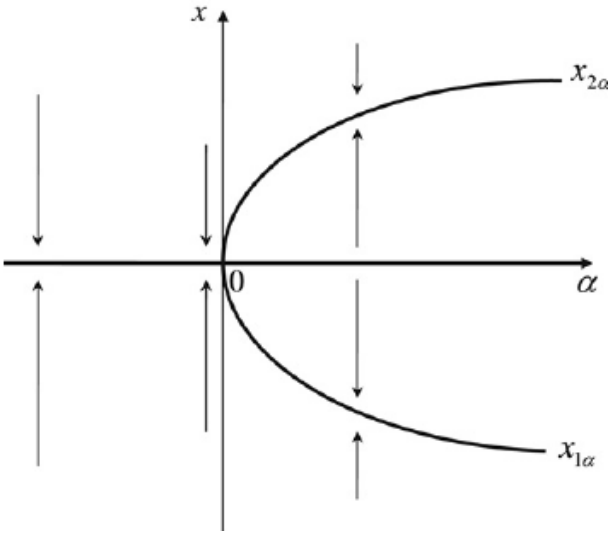


Figure 1.6: The Pitchfork bifurcation diagram.

1.4 Stochastic Processes in Discrete Space and Time

In this section we do an introduction to Stochastic Processes. Of course, this is a very big area of mathematics and we don't have the time to analyze it in its' whole extent. If the reader needs more details about Markov Chains, she must read the book of Norris [27]. Our introduction is based on the notes of Michalis Loulakis [28].

1.4.1 Some basic concepts

Definition 1.4.1. *To define a* ***stochastic process****, we need a set* $\mathbb{X}$*, which is called state space and express all the possible values a stochastic process can take. Moreover, we define a set* T *which includes all the possible times. The simpler way to define our stochastic process, which is defined in a set of times* T *and takes values in a state space* $\mathbb{X}$*, is to think of it as a collection of random variables* $\{X_t\}_{t\in T}$ *that are defined in a probability space* $(\Omega, \mathcal{F}, P)$ *and they take values in* $\mathbb{X}$*.*

Definition 1.4.2. *A stochastic process is called a* ***Markov Chain*** *if the conditional distribution of* X_{n+1} *given* $(X_0, \ldots, X_n)$ *is the same as the conditional distribution of* X_{n+1} *given* X_n*. We can express this definition mathematically as,*

$$P\left[X_{n+1} = y | X_0 = u_0, \ldots, X_{n-1} = u_{n-1}, X_n = x\right] = P\left[X_{n+1} = y | X_n = x\right] \tag{1.18}$$

This simplification allows us to build a beautiful and easy mathematical formulation. The collection of $\mathcal{P} = \{p(x, y)\}_{x,y\in\mathbb{X}}$ is called *transition matrix* of the Markov Chain.

Therefore, we can write this matrix as,

$$\mathcal{P} = \begin{pmatrix} p(u_1,u_1) & p(u_1,u_2) & \dots & p(u_1,u_N) \\ p(u_2,u_1) & p(u_2,u_2) & \dots & p(u_2,u_N) \\ \vdots & \vdots & \ddots & \vdots \\ p(u_N,u_1) & p(u_N,u_2) & \dots & p(u_N,u_N) \end{pmatrix}. \tag{1.19}$$

And if the following two conditions take place,

$$p(x,y) \geq 0,\ \forall x,y \in \mathbb{X}, \qquad \sum_{y\in\mathbb{X}} p(x,y) = 1,\ \forall x \in \mathbb{X} \tag{1.20}$$

then we say that we have a *stochastic matrix*. This matrix helps us to understand how the stochastic process moves from one state to another, however it doesn't say anything about where we start. In this point of view, we must define the *initial distribution* π_0, which must satisfy the following conditions

$$\pi_0(x) \geq 0,\ \forall x \in \mathbb{X}, \qquad \sum_{x\in\mathbb{X}} \pi_0(x) = 1. \tag{1.21}$$

Consequently we can write

$$P\left[X_0 = x_0, \dots, X_n = x_n\right] = \pi_0(x_0)p(x_0,x_1)\dots p(x_{n-1},x_n). \tag{1.22}$$

Due to the fact that we have a Markov Chain, in which each state depends only on the previous one, we can define the probability distributions of each time step as,

$$\pi_{n+1}(y) = P[X_{n+1} = y] = \sum_{x\in\mathbb{X}} P[X_n = x]P[X_{n+1} = y|X_n = x] = \sum_{x\in\mathbb{X}} \pi_n(x)p(x,y) \tag{1.23}$$

or in a simpler form,

$$\pi_{n+1} = \pi_n\mathcal{P}, \quad \forall n \in \mathbb{N}_0. \tag{1.24}$$

Another very important tool is Chapman-Kolmogorov equations that can help us to understand how we multiply stochastic matrices. Given that $\mathcal{P}^{m+n} = \mathcal{P}^m\mathcal{P}^n$, we can write the Chapman-Kolmogorov equations,

$$p^{(m+n)}(x,y) = \sum_{z\in\mathbb{X}} p^{(m)}(x,z)p^{(n)}(z,y), \quad \forall m,n \in \mathbb{N}_0,\ x,y \in \mathbb{X} \tag{1.25}$$

where,

$$p^{(n)}(x,y) = \sum_{x_0\in\mathbb{X}}\sum_{x_1\in\mathbb{X}}\cdots\sum_{x_{n-1}\in\mathbb{X}} \pi_0(x_0)p(x_0,x_1)\dots p(x_{n-1},y). \tag{1.26}$$

1.4.2 The structure of state space

In this section we will do a short discussion about how the state space can be organized in communication classes. So let's see some important definitions.

Definition 1.4.3. *We say that the state $y \in \mathbb{X}$ is accessible from the state $x \in \mathbb{X}$ and we symbolize it us $x \to y$, if there is $n \geq 0$ so that $p^{(n)}(x,y) > 0$.*

Definition 1.4.4. *We say that two states $x, y \in \mathbb{X}$ communicate and we symbolize $x \leftrightarrow y$, if $x \to y$ and $y \to x$.*

Proposition 1.4.1. *The relation $\leftrightarrow$ is an equivalence class, and has the following properties:*

- *Reflexive: $x \leftrightarrow x$, for all $x \in \mathbb{X}$.*
- *Symmetric: $x \leftrightarrow y \Rightarrow y \leftrightarrow x$, for all $x, y \in \mathbb{X}$.*
- *Transitional: $(x \leftrightarrow u$ and $u \leftrightarrow y) \Rightarrow x \leftrightarrow y$, for all $x, y, u \in \mathbb{X}$.*

And knowing the previous definitions, we can also define what is an open, closed and non irreducible class. This is a very fundamental concept to understand the meaning of ergodicity that we need in Monte Carlo methods.

Definition 1.4.5. *An equivalent class $\mathcal{C}$ is called open, if there are states $x \in \mathcal{C}$ and $y \notin \mathcal{C}$ so that $x \to y$. If a class is not open, it is a closed class.*

Definition 1.4.6. *A Markov Chain is called non irreducible, if the whole state space is a closed class.*

1.4.3 Asymptotic behavior of a Markov Chain

As we said before, when we use Monte Carlo Methods, we need ergodicity. Ergodicity is an important feature of our chain because it allows us to estimate the expected values (or statistical moments generally) from our sample.

But before we see the ergodic theorem, it is essential to understand about the invariant distributions of a chain. We symbolize the set of all the possible distributions in $\mathbb{X}$ as $\mathcal{M}(\mathbb{X})$. Thus

$$\pi \in \mathcal{M}(\mathbb{X}) \;\Leftrightarrow\; \pi : \mathbb{X} \to [0,1] \text{ and } \sum_{x \in \mathbb{X}} \pi(x) = 1$$

Now let's give a definition for the invariant distribution.

Definition 1.4.7. *If $\{\pi_n\}_n$ is a sequence of distributions in $\mathcal{M}(\mathbb{X})$, we say that it converges to a distribution $\pi \in \mathcal{M}(\mathbb{X})$ if and only if the probability of π_n for each state convergences to the probability of π for each state respectively. Consequently,*

$$\pi_n \to \pi \in \mathcal{M}(\mathbb{X}) \Leftrightarrow \pi_n(x) \to \pi(x), \quad \forall x \in \mathbb{X}$$

Definition 1.4.8. *If $\{\pi_n\}_n$ is a sequence of distributions of a Markov Chain $\{X_n\}$ and $\pi_n \to \pi \in \mathcal{M}(\mathbb{X})$, then π is the invariant distribution of $\{X_n\}$.*

Another very important concept about Markov Chains is genuine repetitive classes. To give a definition about the genuine repetitiveness, we need to define T_x^+ as the time that is needed for the state x to come back to itself.

Definition 1.4.9. *The state $x \in \mathbb{X}$ is genuine repetitive if the estimated value of the time that it is needed to come back to itself is finite $\mathbb{E}[T_x^+] < +\infty$.*

Remark. *Genuine repetitiveness is a class property, which means that if x is genuine repetitive and $y \in \mathcal{C}_x$, then y is also genuine repetitive.*

This definition means that in genuine repetitive classes, each state is accessible in every time step, and the chain will explore the state space of the class in finite time. At this moment we are ready to see the ergodic theorem, which is basically just the law of large numbers for Markov chains.

Theorem 1.4.1 (Ergodic theorem)**.** *Let $\{X_n\}_{n\in\mathbb{N}_0}$ to be an irreducible, genuine repetitive Markov Chain in a discrete state space $\mathbb{X}$. If $f : \mathbb{X} \to \mathbb{R}$ is a bounded function, then for every initial distribution we have that*

$$P\left[\frac{1}{n}\sum_{k=0}^{n-1} f(X_k) \to \mathbb{E}^{\pi}[f]\right] = 1, \tag{1.27}$$

where $\mathbb{E}^{\pi}[f] = \sum_{x\in\mathbb{X}} f(x)\pi(x)$ is the mean value of the random variable $f(X)$, given that the random variable X follows the invariant distribution of the chain π.

1.5 The Metropolis-Hastings algorithm

Monte Carlo methods are some mathematical computational tools that are based on random numbers generators and help us to simulate probability distributions, or do other computations like estimating integrals or the number π. In our case, we will use them to simulate from the Boltzmann distribution of Ising Model. In this section we discuss about Metropolis-Hastings algorithm, which is one of the most popular Monte Carlo methods.

1.5.1 Defining the problem

Our purpose is to simulate the Boltzmann distribution. But how we define Boltzmann distribution? Suppose that our system is in state μ and $R(\mu, \nu)dt$ is the probability to be in state ν a time dt later. In this manner, $R(\mu, \nu)$ is the *transition rate* for the transition from μ to ν. We also define some weights $w_\mu(t)$ which express the probability that the system will be in state μ at time t. Therefore, we can write the *master equation* which represents the evolution of $w_\mu(t)$,

$$\frac{dw_\mu}{dt} = \sum_{\nu} \left[w_\nu(t)R(\nu,\mu) - w_\mu(t)R(\mu,\nu)\right], \tag{1.28}$$

where

$$\sum_{\mu} w_\mu(t) = 1 \tag{1.29}$$

From the theory of dynamical systems, we know that to define an equilibrium state, the term on the right-hand side of eq. 1.28 must be zero. If this happens it means that $dw_\mu/dt = 0$ and the weights remain constant over time. Also the behavior of $R(\mu, \nu)$ will change. Because having reached the equilibrium, they must take only values that arise out of the thermal nature of our system. Therefore, we define the *equilibrium occupation probabilities* as,

$$\pi_\mu = \lim_{t\to\infty} w_\mu(t). \tag{1.30}$$

And Gibbs proved that if the system is in thermal equilibrium with a reservoir at temperature T, then

$$\pi_\mu(t) = \frac{1}{Z} e^{-E_\mu/kT} \tag{1.31}$$

where E_μ is the energy of state μ and $k = 1.38 \times 10^{-23}\ JK^{-1}$ is the Boltzmann's constant. This is the Boltzmann distribution. Additionally, we denote the quantity $(kT)^{-1}$ with the symbol β. The constant or the partition function is

$$Z = \sum_\mu e^{-E_\mu/kT} = \sum_\mu e^{-\beta E_\mu} \tag{1.32}$$

Thus what we want is to simulate from this distribution and to be able to compute quantities like

$$\langle Q \rangle = \sum_\mu Q_\mu \pi_\mu = \frac{1}{Z} \sum_\mu Q_\mu e^{-\beta E_\mu} \tag{1.33}$$

as it is the magnetization that we defined in the first section of this chapter. In our case, we also know the Hamiltonian of our system (eq. 1.1) and this is very helpful, because we know from physics that every system tends to minimize its' energy. Metropolis-Hastings algorithm uses this observation to simulate from Boltzmann distribution.

But why we have to work with Metropolis-Hastings simulation instead of using other Monte Carlo methods? Ok, let's assume that we try to simulate by using the popular Monte Carlo algorithm of rejection. So what we do is to choose states by accepting or rejecting them with a probability proportional to $e^{-\beta E_\mu}$. The problem is that the probability for their acceptance would be exponentially small, and we would end up rejecting all the states. This means that we cannot simulate in this way and we must try something else.

1.5.2 The requirements of the algorithm

To start working with Metropolis-Hastings algorithm, three requirements must be satisfied. Once we have them, we can simulate from any probability distribution we want (however now we focus on Boltzmann distribution). These are the following ones:

1. **The assumption of Markov process:** If we are in a state μ, we can go th the state ν with probability $p(\mu, \nu)$. These probabilities $p(\mu, \nu)$ should now vary over time, and they must depend only on the current states μ and ν.

2. **The assumption of ergodicity:** The condition of ergodicity is the requirement that our system must be able to reach any state, if we run it for long enough.

3. **The assumption of detailed balance:** This condition ensures that that it is the Boltzmann distribution which we generate after our system has come to equilibrium, rather that any other distribution. This assumption is expressed mathematically as,

   $$\pi_\mu p(\mu, \nu) = \pi_\nu p(\nu, \mu). \tag{1.34}$$

   And in case of Boltzmann distribution we have that

   $$\frac{p(\mu, \nu)}{p(\nu, \mu)} = \frac{p_\nu}{p_\mu} = e^{-\beta(E_\nu - E_\mu)}. \tag{1.35}$$

   This equation together with $\sum_\nu p(\mu, \nu) = 1$ are the two constraints we need to have a Boltzmann distribution.

1.5.3 The recipe

First of all, we break the probability in eq. 1.35 in two parts,

$$p(\mu,\nu) = g(\mu,\nu)A(\mu,\nu) \tag{1.36}$$

The quantity $g(\mu,\nu)$ is called *selection probability* and the quantity $A(\mu,\nu)$ is called *acceptance ratio*. The reason why we do this is because we don't know which are the correct Boltzmann probabilities and thus we have not any information about how we will define the transition probabilities. The former quantity tell us with which probability we can go from the state μ to ν, but the latter tell us with which probability we accept this choice. In this manner, we can write

$$\frac{p(\mu,\nu)}{p(\nu,\mu)} = \frac{g(\mu,\nu)A(\mu,\nu)}{g(\nu,\mu)A(\nu,\mu)} \tag{1.37}$$

Therefore, in our case we choose:

$$A(\mu,\nu) = \begin{cases} e^{-\beta(E_\nu - E_\mu)}, & \text{if } E_\nu - E_\mu > 0 \\ 1, & \text{otherwise} \end{cases} \tag{1.38}$$

Assuming that in our lattice we have N spins, then each one has equal probability to flip. Thereafter, we can determine our selection probabilities,

$$g(\mu,\nu) = \frac{1}{N} \tag{1.39}$$

and so on we can write our detailed balance as

$$\frac{p(\mu,\nu)}{p(\nu,\mu)} = \frac{g(\mu,\nu)A(\mu,\nu)}{g(\nu,\mu)A(\nu,\mu)} = \frac{A(\mu,\nu)}{A(\nu,\mu)} = e^{-\beta(E_\nu - E_\mu)}. \tag{1.40}$$

From this equation it is clear that the acceptance ratio we chose in eq. 1.38 it is well defined, since it satisfies the condition of detailed balance.

Now let's come back to the Ising model. We have N^2 spins in our lattice, where each one of them can take the values ± 1. A very helpful assumption we can do, is this of periodic conditions. We assume that even if we work in a finite lattice, there is not any boundary. Our lattice boundaries are joined together, so as if we try to get out of the lattice from the right side, we will end up in the left side. To picture this kind of boundary conditions, we can imagine something like the Fig. 1.7.

Assuming that $B = 0$, we can also compute the energy difference, when we flip one spin of our lattice. It is very easy to do this straightforward calculation and write,

$$\begin{aligned} E_\nu - E_\mu &= -J \sum_{<ij>} s_i^\nu s_j^\nu + J \sum_{<ij>} s_i^\mu s_j^\mu \\ &= -J \sum_{i \text{ n.n. to } k} s_i^\mu (s_k^\nu - s_k^\mu) \end{aligned} \tag{1.41}$$

The important observation here is that because our spins can take only two values ± 1, we can write,

$$s_k^\nu - s_k^\mu = -2s_k^\mu \tag{1.42}$$

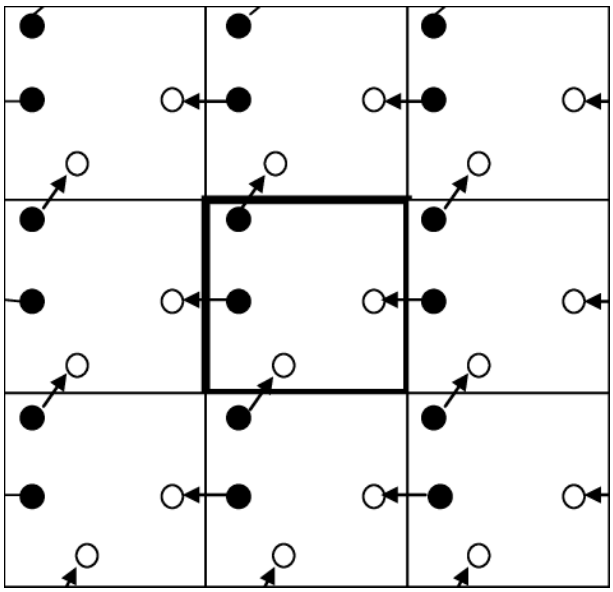


Figure 1.7: Periodic boundary conditions.

Thus the energy difference is,

$$E_\nu - E_\mu = 2J \sum_{i \text{ n.n. to } k} s_i^\mu s_k^\nu = 2J s_k^\mu \sum_{i \text{ n.n. to } k} s_i^\mu. \tag{1.43}$$

At this point we already have described the Metropolis-Hastings algorithm. There is only one thing we haven't discuss and this is the initialization of the algorithm. But to be fair, the concept of initialization doesn't play a very important role, because if we have constructed the algorithm correctly, our chain will reach the equilibrium no matter what initial configuration we chose. However, there are two ways to initialize our lattice: *cold and hot*. In cold configuration all spins look at the same direction, whereas in hot configuration the spins are random.

Sometimes, it is important to have a simple picture of what we do. This is the reason why we summarize the recipe of Metropolis-Hastings algorithm in the following steps.

Metropolis-Hastings algorithm:

1. We create a $n \times n$ lattice with hot or cold initial configuration.
2. For the purposes of our Monte Carlo simulation we start off by randomly flipping a spin and calculate the change in energy, ΔE.
3. If ΔE is negative, we accept the new configuration and move to the next step.
4. If ΔE is positive, we select a random number between 0 and 1, and accept the configuration only if the number is less than $e^{-\frac{\Delta E}{kT}}$.
5. Repeat the last three steps.

"The Three Laws of Robotics:
1: A robot may not injure a human being or, through inaction, allow a human being to come to harm;
2: A robot must obey the orders given it by human beings except where such orders would conflict with the First Law;
3: A robot must protect its own existence as long as such protection does not conflict with the First or Second Law;
The Zeroth Law: A robot may not harm humanity, or, by inaction, allow humanity to come to harm."

- Isaac Asimov, I, Robot

2 Machine Learning

In this thesis we tried to combine the Ising model with Machine Learning. Consequently, it is important to know the fundamental terminology of Machine Learning, to have an idea about how it can be combined with Complex Systems problems and discuss the common points of these two fields.

2.1 Machine Learning perception

Machine Learning is the science (or the art) of finding patterns. Its' main purpose is to build machines that can recognize and categorize data. Therefore, there are two main categories of problems in Machine Learning: *classification*, where we are trying to classify data in some categories, and *regression*, where the output is a continuous variable. An example of a classification problem is handwritten digit recognition, whereas an example of regression is when we have some experiment data and try to find a function that relate some of the measured variables. Regression is a very common problem in physics. There are three main ways that our algorithm learn the data: *supervised, unsupervised and reinforcement*. In supervised learning, there are some labels in our data and the algorithm finds patterns based on this labels. On the other hand, in unsupervised learning there are not any labels and the algorithm finds patterns in a more agnostic way. Reinforcement learning (RL) is an area of machine learning concerned with how intelligent agents ought to take actions in an environment in order to maximize the notion of cumulative reward ([60]).

2.2 A simple example of classification

Here we present a simple example of classification from the book of Duda and Hart [9] just to become familiar with this type of Machine Learning. Let's assume that we have a problem, where we have images from two types of fish: sea bass and salmon. In these terms, there are some characteristics that help us to do this kind of classification, like length, lightness, width, number and shape of fins etc., but on the other hand we have noise and variations of the images. Our goal is to use the more

information we can for the former type of characteristics and eliminate the noise as much as possible.

Therefore, there is always a kind of *pre-processing and feature extraction* of our data, which means that we should do some operations (like converting images to black and white or to separate fish from background) and then we should think which of these features we have for our data are the most important ones to describe our problem. There is a great variety of feature extraction techniques (like choosing features with minimum correlations or stepwise eliminations), however in this thesis we have a problem with simulation data, and thus we based the feature extraction mainly in our physical intuition. Consequently, we have a procedure like this that it is described in Fig. 2.1. Sometimes we can change our pre-processing and feature extraction having done the classification to see if we have better results.

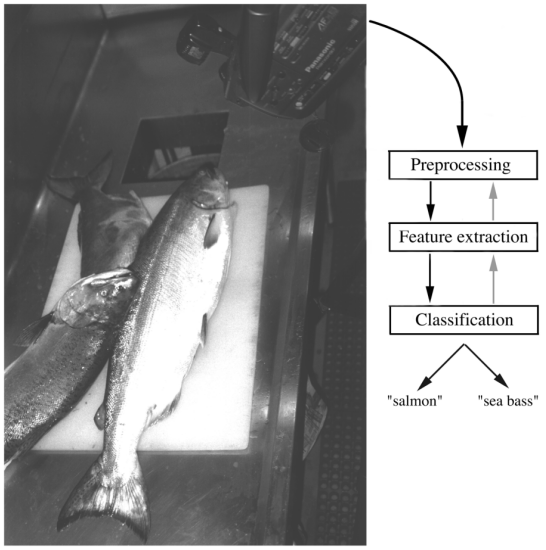
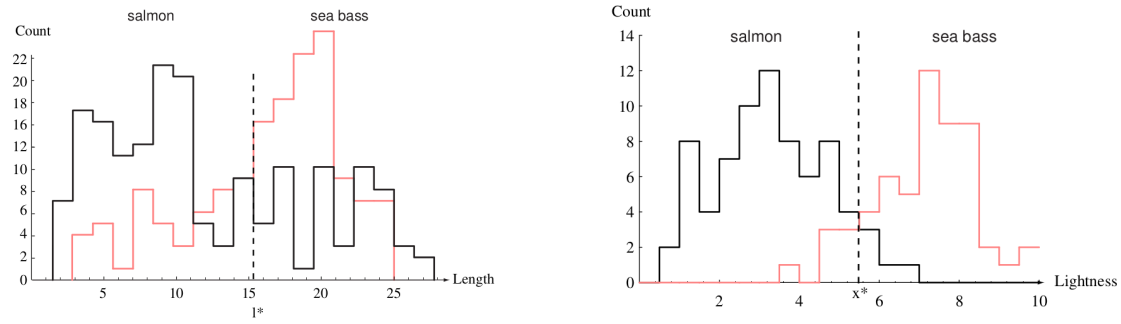


Figure 2.1: The strategy we should follow for the example with fish of [9].

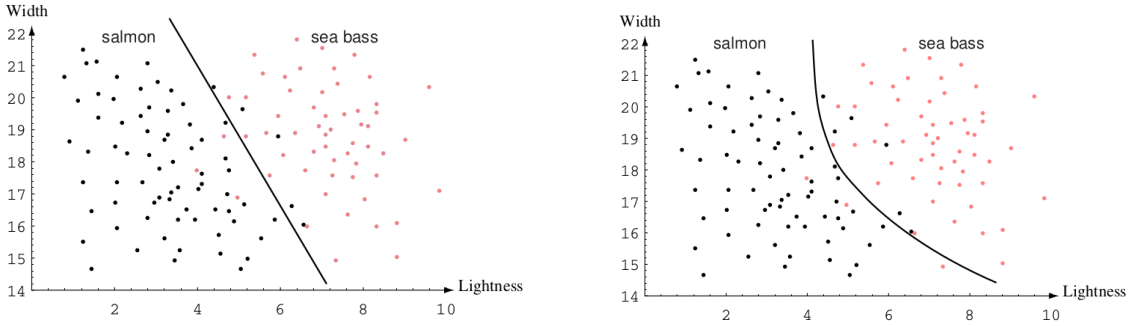
The main goal is to keep the minimum number of features that can separate these two types of fish. Having done that, our classifier will fit two probability distributions for each class. They way we fit these probability distributions is the Machine Learning algorithm we use. At this point there are two ways to solve our classification problem: we can either predict the probability for a specific image of fish to be in a specific class, or we can categorize it in a binary way (yes or no). The most user-friendly way to solve our problem is the second one, in which we must choose a specific threshold l^* with which we would take our decision (Fig.2.2).

In Fig. 2.3 we can see three types of decision boundaries in the space of lightness and length. The main goal of a Machine Learning algorithm must be to find the optimal decision boundary, viz the decision boundary that it is able to discriminate better the classes of our problem. The first kind of fitting is a linear curve 2.3a, which is able to classify a lot of our sample but it is not flexible enough to reassure that it will have the same good behavior for a bigger amount of data. The third curve 2.3c is a very flexible curve that learns a lot of detail for our decision boundary. This is another behavior we usually avoid because it makes also our algorithm very bad in generalization. Finally, we need a model like this in 2.3b, which is a balance between a very simple and a very complex model.

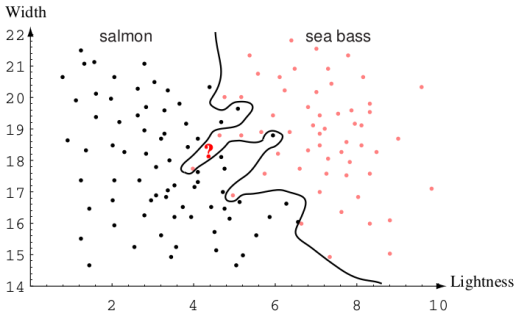


(a) (b)

Figure 2.2: The probability distribution of fish for each class: salmon and sea bass. In these diagrams we put a threshold value l^* (or decision boundary) which leads to the smaller number of errors on average. We have chose length and lightness as the most important features for this classification problem.



(a) Underfitting (b) Optional model



(c) Overfitting

Figure 2.3: Choosing a decision boundary for the problem of fish classification.

2.3 Bayesian Decision Theory

Supposing that we have some classes $\mathcal{C}_k$ with known a-priori probabilities $p(\mathcal{C}_k)$ and some data $\boldsymbol{x}$, then the Bayes rule can be expressed in the form,

$$p(\mathcal{C}_k|\boldsymbol{x}) = \frac{p(\boldsymbol{x}|\mathcal{C}_k)p(\mathcal{C}_k)}{p(\boldsymbol{x})}. \tag{2.1}$$

This is the general expression of Bayes rule. However, in Machine Learning our goal is to minimize the missclassification error. Therefore, we need a rule that assigns each value of $\boldsymbol{x}$ to one of the available classes. This rule will divide our input space

into some *decision regions* $\mathcal{R}_k$, and the boundaries between these regions are called *decision boundaries* or *decision surfaces*. In case we have only two classes, we can write

$$\begin{aligned} p(\text{mistake}) &= p(\boldsymbol{x} \in \mathcal{R}_1, \mathcal{C}_2) + p(\boldsymbol{x} \in \mathcal{R}_2, \mathcal{C}_1) \\ &= \int_{\mathcal{R}_1} p(\boldsymbol{x}, \mathcal{C}_2) d\boldsymbol{x} + \int_{\mathcal{R}_2} p(\boldsymbol{x}, \mathcal{C}_1) d\boldsymbol{x} \end{aligned} \tag{2.2}$$

In this manner, we assign $\boldsymbol{x}$ in the class that eq. 2.2 is minimum. For instance, if $p(\boldsymbol{x}, \mathcal{C}_1) > p(\boldsymbol{x}, \mathcal{C}_2)$ then the given value of $\boldsymbol{x}$ is assigned to the class $\mathcal{C}_1$. In Fig. 2.4 we can see that the best way to minimize missclassification error is by choosing the regions so that the boundary point $\hat{x}$ to be the point that $p(\boldsymbol{x}, \mathcal{C}_1) = p(\boldsymbol{x}, \mathcal{C}_2)$. If we choose any other point as boundary, the area under curves becomes bigger, and so on the missclassification error. The point where the two joint probabilities become equal, is called *Bayesian decision point*. Nevertheless, we don't always want to have the same error for both of our classes. A very good example about this, are Covid-19 tests. In order to be sure that we will find the infected people we maximize the probability somebody to be positive given that she hasn't caught the virus, and minimize the probability that is negative given that she has Covid-19. This is due to the fact that it is more dangerous if somebody has the virus and doesn't know it, than has not the virus and the test is positive, she simply will do a second test.

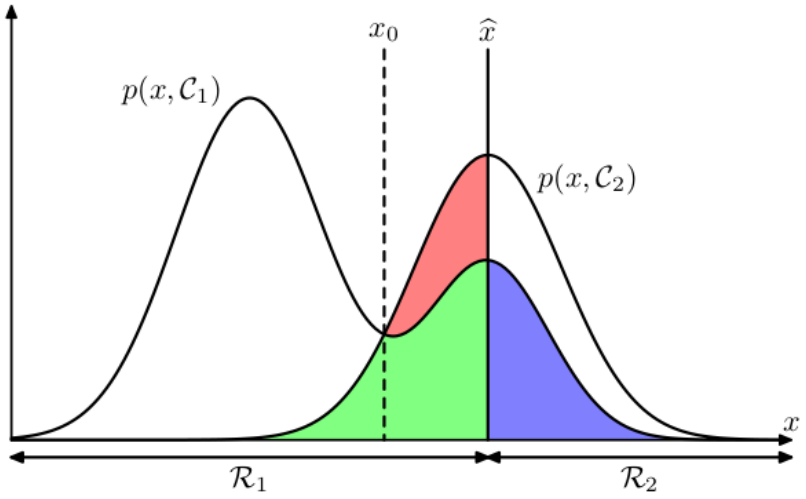


Figure 2.4: A schematic illustration that explains the minimization of missclassification error. ([8])

Bayesian theory is very important in Machine Learning. However, in this thesis we work mainly with regression problems. Therefore, this paragraph about Bayesian theory exists only to have a basic intuition.

2.4 A simple example of regression

In this thesis we will work with a regression problem. Therefore, it is essential to have an initial perceptive and try to understand a very basic example of polynomial curve fitting from the book of Bishop [8]. In this example, we introduce the reader to the main ideas and philosophy of Machine Learning.

Let's assume that we have a training set of N observations $\boldsymbol{x} = (x_1, x_2, \ldots, x_N)^T$ and a set of target values $\boldsymbol{t} = (t_1, t_2, \ldots, t_N)^T$, but we don't know it. So our data look like the Fig. 2.5. These target variables are produced by a function $f(x) = \sin(2\pi x)$. Our goal is to learn our training set so as to make predictions for a variable $\hat{t}$.

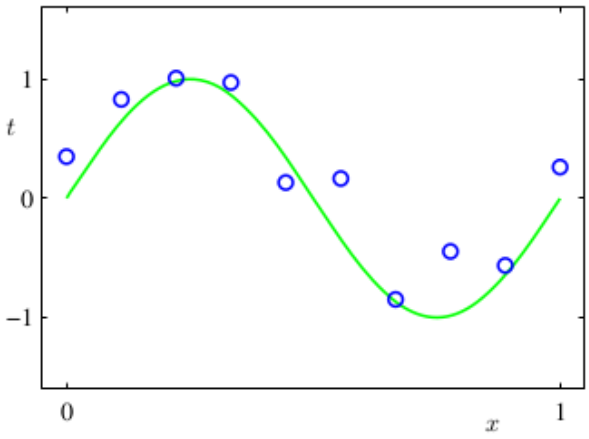


Figure 2.5: The dataset of our example [8].

Due to the fact that we don't know the real function that lurks behind our data, we shall fit the data using a polynomial function of the form,

$$y(x, \boldsymbol{w}) = w_0 + w_1 x + w_2 x^2 + \cdots + w_M x^M = \sum_{j=0}^{M} w_j x^j \tag{2.3}$$

where M is the order of polynomial and $\boldsymbol{\phi}(\cdot)$ is a polynomial transformation. In this problem we want to determine the coefficients $\boldsymbol{w} = (w_0, w_1, w_2, \ldots, w_M)^T$. To do that we use a *loss function*, which will help us to find the coefficients of the polynomial by computing its' derivatives and minimizing it. There is a great variety of loss functions and play a very important role in Machine Learning. In this thesis, the whole second chapter analyzes only one loss function. For the problem of polynomial regression, this loss function is Mean Square Error (MSE),

$$E(\boldsymbol{w}) = \frac{1}{2} \sum_{n=1}^{N} (y(x_n, \boldsymbol{w}) - t_n)^2 = \sum_{i=1}^{N} \epsilon_i^2 = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \tag{2.4}$$

To find the coefficients, we must solve the minimization problem,

$$\frac{\partial E(w)}{\partial w_i} = \sum_{i=1}^{N} (y(x_n, w) - t_n)\, x_n^i = \sum_{n=1}^{N} \left\{ \sum_{j=0}^{M} w_j x_n^j - t_n \right\} x_n^i = 0. \tag{2.5}$$

In case of polynomial regression it is easy to calculate these coefficients, but generally speaking it is not always easy to find closed forms for the model parameters in Machine Learning.

Another question we shall answer is about the order of polynomial we shall use. It is not that every polynomial can fit our data. For example, if we see Fig. 2.6, when $M = 0$ our model is a line. This model is a very poor model to describe our data since

we cannot predict anything. This is an example of *underfitting*. If $M = 3$ we have a better description of our data. We have a relatively simple model and we can do predictions with it. However, in case that $M = 9$, our model learns too many details of our data. This is also something we try to avoid, and it is a paradigm of *overfitting*.

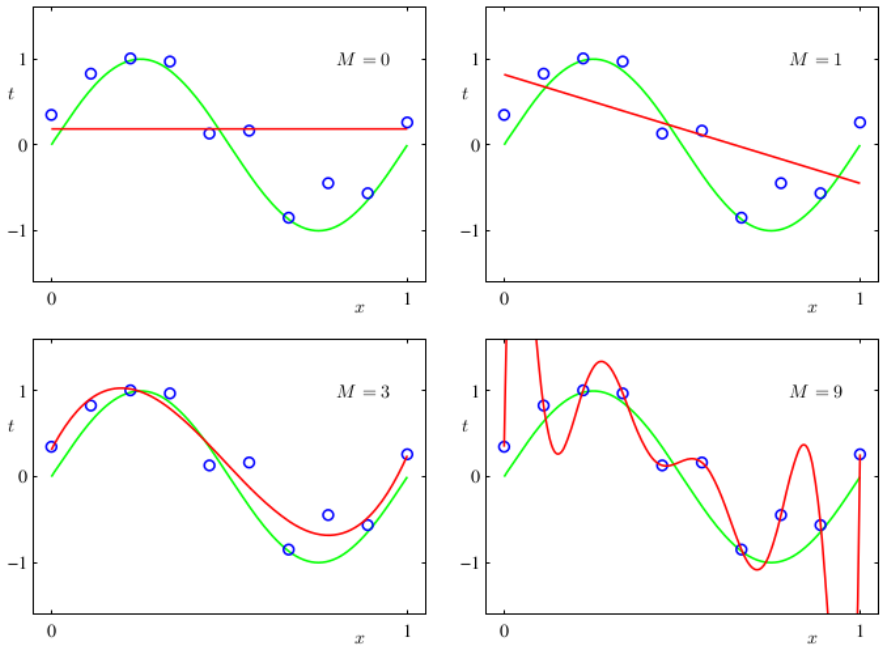


Figure 2.6: How different order polynomial (red curve) fit our data, in contrast to the sinusoidal function (green curve) that generated them. ([8])

We usually put a penalty term in our loss function so that to choose the simpler model that describes our data,

$$\tilde{E}(\boldsymbol{w}) = \frac{1}{2}\sum_{i=1}^{N}(y(x_n, \boldsymbol{w}) - t_n)^2 + \frac{\lambda}{2}||\boldsymbol{w}||^2 \tag{2.6}$$

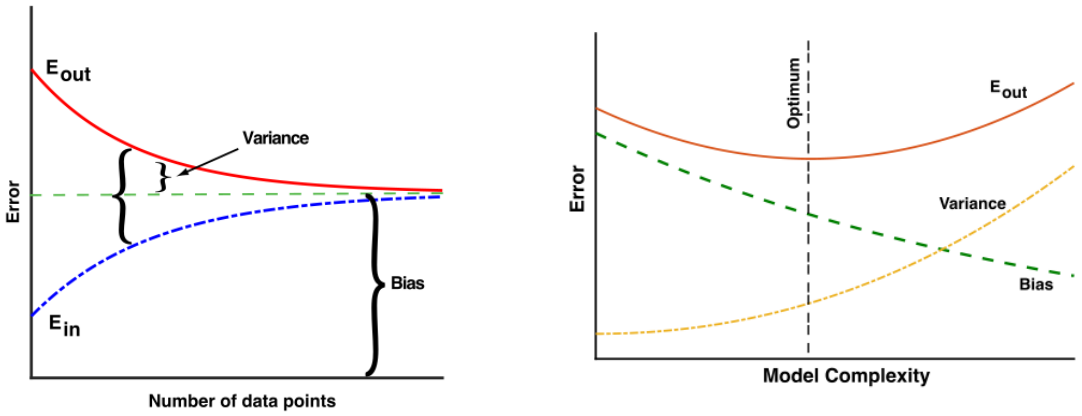
where $||\boldsymbol{w}||^2 = \boldsymbol{w}^T\boldsymbol{w} = w_0^2 + w_1^2 + \cdots + w_M^2$, and the coefficient λ governs the relative importance of the regularization term compared with the sum-of-squares error term.

From this example, we learn that we must always search for a balance between simplicity and complexity of our model. We don't want models, which are too complicated because it is difficult to generalize them in other datasets, but on the other hand we don't want models that are too simple and give us a poor representation of our data. This is what is called 'bias-variance trade-off': *"The simpler model has more bias but is less dependent on the particular realization of the training dataset, i.e. less variance."* ([24]). *Generalization* is a very important term in Data Science and it is one of the main issues that a data scientist has to struggle with.

2.5 Bias-Variance tradeoff

To be more specific, there are two main factors that can affect the efficiency of a Machine Learning algorithm: the complexity of model and the size of dataset. Therefore, in Fig. 2.7a there is a diagram that represent how the error of train and

test set[1] changes as a function of number of points of our dataset. It is clear that the variance (viz. the difference between train and test error) becomes minimum when we have a lot of data. However, it is not always easy to have that many data. On the other hand, in Fig. 2.7b we see how test error, bias and variance depend on the model complexity. The main goal is to choose a model that do not have neither high bias-variance, nor high test error.



(a) Error versus size of dataset.

(b) Error versus model complexity.

Figure 2.7: These two figures explain bias-variance tradeoff. Here E_{in} is training error and E_{out} is generalization error ([24]).

2.6 The curse of dimensionality

In our example with polynomial regression the things were easy because we had only one dimension in our problem. However, one of the most common issues in Machine Learning is this of dimensionality. Commonly, we have very big datasets with many features to learn and there is not any way to illustrate them. Apart from that, there is a more important problem than the illustration of datasets, and this is that if we increase the dimension is very difficult to ensure that our data cover all the cells of our space (Fig. 2.8). The cells increase exponentially with the dimension of our problem and thereby the size of our data also increases exponentially.

To solve this problem, there are methods to discard correlated features such as stepwise techniques, or methods that help us to downgrade the dimension of features combining some principal components (i.e. Principal Component Analysis or Linear Discriminant Analysis).

2.7 Complex Systems and Machine Learning

2.7.1 History

Complex Systems and Machine Learning are two fields that look different. The former is related to theoretical models and offer a variety of mathematical tools that

[1] In Machine Learning we usually split our data in two groups: train and test. We fit our model in train set and compute the accuracy score, and then we do predictions in test set to see how good is our classifier or regressor in generalization.

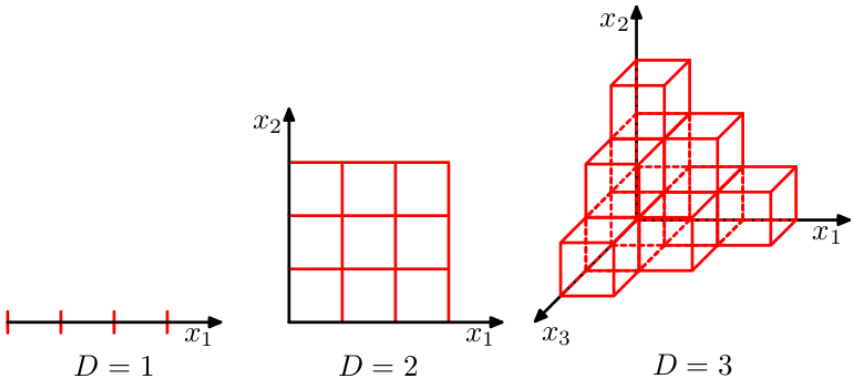
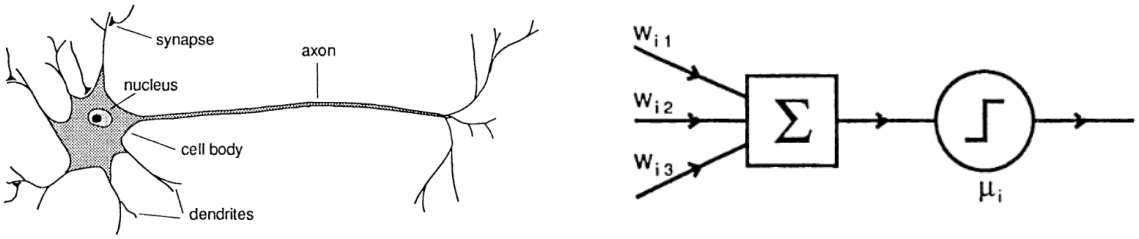


Figure 2.8: Illustration of the curse of dimensionality, showing how the number of regions of a regular grid grows exponentially with the dimensionality D of the space. For clarity, only a subset of the cubical regions are shown for $D = 3$ ([8]).

can help us to understand any aspect of science: astrophysics, complex networks, sociology, neuroscience, nanotechnology, solid state physics, non linear waves, traffic flow or even quantum mechanics, and the later is about data-driven techniques that help us to learn patterns from our data.



(a) A biological neuron.

(b) An artificial neuron.

Figure 2.9: A biological versus an artificial neuron. ([2])

In this thesis we work with a simulation problem. There are no data from any experiments, however we can produce numbers by running simulations, and then feed them to Machine Learning regressors. Therefore, this is the first kind of interaction: to use Machine Learning to understand problems of Complex Systems (or physics generally). Another aspect of this interaction, is that Complex Systems and physics can help us to understand Machine learning better. The first neural network models started from the terminology of physics, and the struggle of giving an explanation about how human brain function. In Fig. 2.9 we can see a biological neuron that exists in our brain in contrast to an artificial neuron that we use to build neural networks in machine learning. The connection between these two representation is a kind of loose, but the second mathematical model was inspired from the first one. At this moment, we don't understand why artificial networks are trained or work generally, and this is an open problem for research.

The first neural models where based in the energy minimization principle of physics, and thus they were named *energy models*. We can imagine the energy of our network as a surface in a multidimensional space like this in Fig. 2.10. However, this representation is not always easy because we have to deal with too many network parameters.

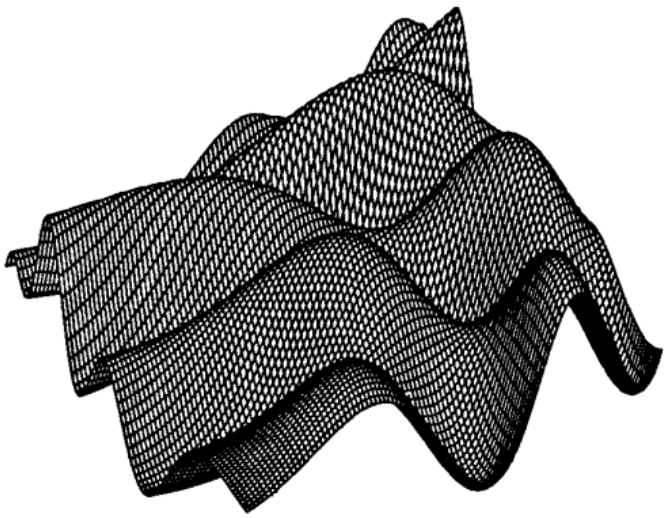


Figure 2.10: We can think our energy as a surface in a multidimensional space, where z-axis is our energy and in x-y space are the parameters of our network. ([22])

2.7.2 Common terminology

Both of these fields use some common terminology, but it is essential to make clear which are the exact differences between the two fields. Here we focus in three terms that are common in Complex Systems and Machine Learning,

1. **Complexity:** When we speak about complexity in Machine Learning, we mainly mean the complexity of our model. How many parameters, neurons or layers or layers we use in our model is model's complexity. On the other hand, in Complex Systems, complexity is a state between the total randomness and symmetry (Fig. 2.11).

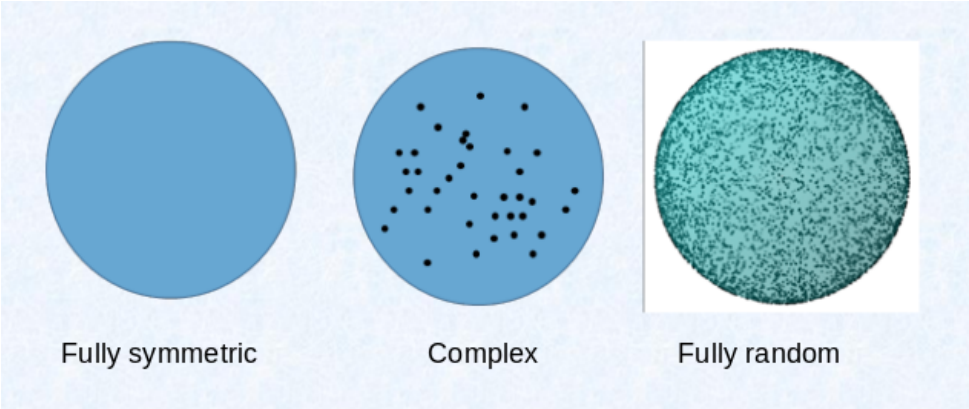


Figure 2.11: Complexity in Complex Systems.

2. **Entropy:** In Machine Learning usually when we speak about entropy, we mean information entropy. Our measure of information content will depend on the probability distribution $p(x)$, and we therefore look for a quantity $h(x)$ that is a monotonic function of the probability $p(x)$ and that expresses the information content,

$$h(x) = -\log_2 p(x) \tag{2.7}$$

The form of $h(\cdot)$ can be found by noting that if we have two events x and y that are unrelated, then the information gain from observing both of them should be the sum of the information gained from each of them separately, so that $h(x, y) = h(x) + h(y)$.

The average amount of information that they transmit in the process is obtained by taking the expectation of the previous equation with respect to the distribution $p(x)$ and is given by

$$H[x] = -\sum_x p(x) \log_2 p(x) \tag{2.8}$$

In case of Complex Systems, we use the definition of Statistical Mechanics. According to the source [23], if a system can be at any of $i = 1, 2, \ldots, W$ states with probability p_i, its entropy is given by the famous formula

$$S = -k \sum_{i=1}^{W} p_i \ln p_i \tag{2.9}$$

At thermal equilibrium, the probabilities that optimize the entropy subject to $\sum_i p_i = 1$, given the energy spectrum E_i and temperature T of these states are:

$$p_i = \frac{e^{-\beta E_i}}{Z}, \quad Z = \sum_{i=1}^{W} e^{-\beta E_i} \tag{2.10}$$

where $\beta = 1/kT$ and Z is the so-called partition function.

3. **Correlation:** In Machine Learning, correlation is a statistical measure that expresses the extent to which two variables are linearly related (meaning they change together at a constant rate). Its a common tool for describing simple relationships without making a statement about cause and effect. Usually, we use the sample correlation coefficient, r, quantifies the strength of the relationship. Correlations are also tested for statistical significance.

   In Complex Systems to measure correlations, we usually use correlation functions. The most common definition of a correlation function is the canonical ensemble (thermal) average of the scalar product of two random variables, $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, at positions R and $R + r$ and times t and $t + \tau$:

$$C(r, \tau) = \langle \mathbf{s}_1(R, t) \cdot \mathbf{s}_2(R + r, t + \tau) \rangle \; - \langle \mathbf{s}_1(R, t) \rangle \langle \mathbf{s}_2(R + r, t + \tau) \rangle \, . \tag{2.11}$$

   Here the brackets, $\langle \ldots \rangle$, indicate the above-mentioned thermal average. It is a matter of convention whether one subtracts the uncorrelated average product of $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$, $\langle \mathbf{s}_1(R, t) \rangle \langle \mathbf{s}_2(R + r, t + \tau) \rangle$ from the correlated product, $\langle \mathbf{s}_1(R, t) \cdot \mathbf{s}_2(R + r, t + \tau) \rangle$, with the convention differing among fields.

2.7.3 Research

Even if some physicists have doubts about the applications of Machine Learning in physics or Complex Systems, the interaction of these two fields is a very serious subject with a lot of research, which offers a rich collection of problems. In the review

article [4] there is a very detailed discussion about applications of Machine Learning in Complex Systems.

A big amount of these applications concerns Reservoir Machines, which are Neural Networks that can solve a great variety of non-linear dynamical systems problems, like forecasting chaotic systems [39], detecting unstable periodic orbits [40], synchronization problems [41] or even more practical problems like applications on semiconductors [42]. Another application of this kind is this of the paper [43], where the authors apply sequence to sequence learning to predict spatiotemporal systems of non-linear waves or [44] where the authors use classic neural networks to predict extreme events in Henon map.

On the other hand, there are some works that focus on applications of Machine Learning and Complex Systems. Such works are [45] where the authors use k-Nearest Neighbors, Support Vector Machines, Feed-forward Networks and Reservoir Computing to predict the amplitude of chaotic laser pulses, or [46] where the authors predict periodic and chaotic time series of Hindmarsh-Rose model of neural dynamics using Reservoir Computing.

Moreover, apart from this kind of research that Machine Learning is used to predict the behavior of very complex theoretical models, there are also a lot of works with data. Examples from this domain of research are [47] where the authors use network similarity and network embedding algorithms to do link predictions in real networks, or [48] where the authors propose an algorithm alternative to Convolutional Neural Networks which is useful for biomedical diagnosis, or [49] where the authors create networks from the light curves of different binary stars, and use the parameters of the network to do a classification for the different kinds of binary stars systems. Another interesting application is on the paper [50] where the authors work with both simulation and data. Initially, they extract the epidemic networks of SIS model and with a dimensionallity reduction algorithm, they convert the network into an image. After that, they combine structural and dynamical information of the network and feed it to a Convolutional Neural Network with confusion scheme evaluation.

Regarding to Ising Model specifically, there is a lot of work which is related to the reconstruction of its phase transitions and the determination of critical point. For example, in [51] and [54] the authors use autoencoders and dimensionality reduction techniques (such as Principal Component Analysis) to reconstruct phase transitions of the Ising Model. Another direction of research is this of Boltzmann Machines. Boltzmann Machines attracted the attention and curiosity of many scientists because of critical point slowing down phenomenon. If we train a Boltzmann Machine with Ising model lattices in different temperatures, it always learns the critical point. The reason why this happens is an open problem and there is a lot of theoretical and computational work behind these algorithms. Some of these works are in [34], [35], [36], [37]. This strange behavior of Boltzmann Machines was something that really attracted the interest of the author of this thesis.

Mathematics is the art of giving the same name to different things.

- J. H. Poincare

3 Rough Surfaces Simulation

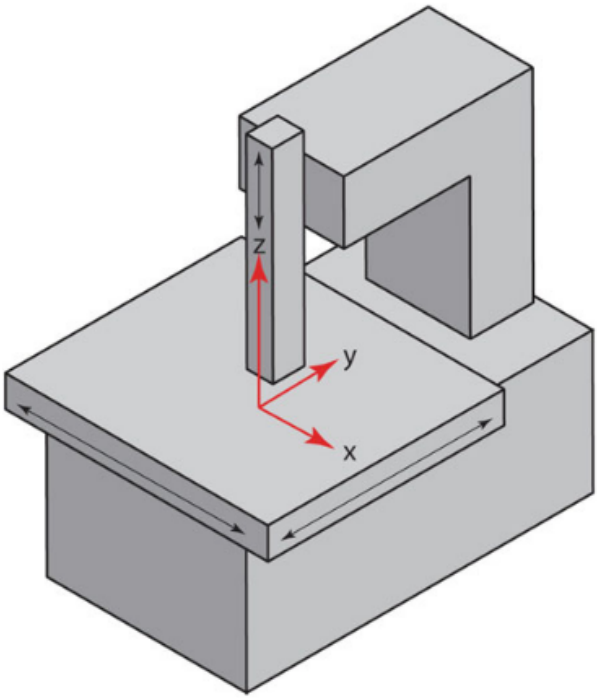
In this chapter we investigate the idea of rough surfaces simulation. Our main goal is to combine this kind of simulation to reach faster the equilibrium of Metropolis-Hastings algorithm with Machine Learning. Nevertheless, before we do that we need to understand what are rough surfaces, with which statistical parameters can be described and how we can produce them with a kind of simulation program. Therefore, in this chapter we focus in these questions to be prepared for the next chapter, where we will create an algorithm that will be able to reconstruct Ising model and its dynamical behavior.

3.1 Surface Topography

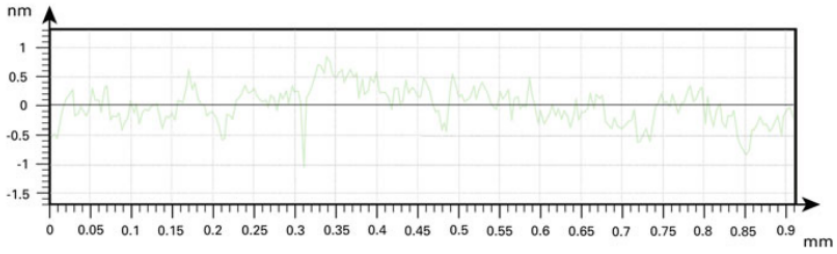
According to [7], the surface is usually defined as the feature on a component or device that interacts with the environment in which the component is housed or in which the device operates, or with another surface. The characterisation of surface topography is a complex branch of metrology with a large range of parameters. The proliferation of surface texture characterisation parameters has been referred to as parameter rash since there can be over one hundred parameters to choose from. However, in this thesis we don't need to do an extensive investigation of these parameters, we will just use some of the most common ones and we will pass by.

The most important factor of surface topography is the height function $z(x)$ which is produced by measuring a line across the surface. Usually, when we characterize a surface we work with a right-handed Cartesian set in which x axis is the direction of the line and y axis lies nominally in the real surface. Of course, z is the outward direction from the material to the surrounding medium. We can see the Cartesian coordinate system in Fig. 3.1a.

The measurement is carried out with a stylus by traversing it across a line of the surface. The profile is usually extracted by an areal optical instrument. When making measurements with a stylus instrument, the traversing direction for assessment purposes is defined as perpendicular to the direction of the lay, unless otherwise indicated. An example of a result of this kind of measurement is in Fig. 3.1b.



(a) Coordinate system for profile measurement.



(b) Example of the result of a profile measurement.

Figure 3.1: An example of a coordinate system and a result of the profile measurement.

3.2 Parameters that Characterize Rough Surfaces

At this point we need to have a basic idea about the parameters that can characterize rough surfaces. In this section we present some of these parameters as they were introduced in the master thesis of A. Stellas [19].

3.2.1 Statistical Moments

Let's assume that we have a probability distribution function $p(x)$ which is positively defined and normalized. We can define a probability measure in the interval $(a, b]$ as,

$$\int_a^b p(x)dx = P(a < X \leq b) \tag{3.1}$$

where X is a random variable with values x. Generally speaking, the n-th moment u_n of the probability distribution $p(x)$ near the mean value $\bar{x}$ is defined as,

$$u_v = \mathbb{E}\left[(x - \bar{x})^v\right] = \int_{-\infty}^{+\infty} (x - \bar{x})^v p(x)dx \tag{3.2}$$

The first order statistical moment is what we call mean value, and it is defined as,

$$\bar{x} = \mathbb{E}[X] = \int_{-\infty}^{+\infty} xp(x)dx \tag{3.3}$$

The second order moment is called variance $u_2 = \sigma^2$, and the third order is a measure of assymetry of our distribution and it is called skewness. Its normalized form is $Sk = u_3/\sigma^3$. The fourth order moment is called kurtosis $Ku = u_4/\sigma^4$, and it is a measure about how sharp is our distribution. There is a restriction for the last two moments $Ku - Sk^2 - 1 \geq 0$ (Fig. 3.2).

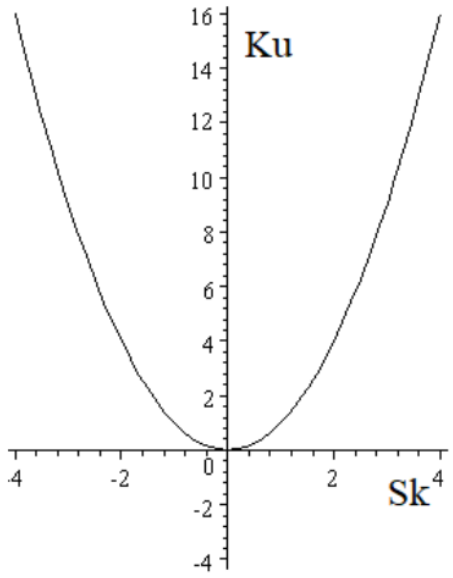
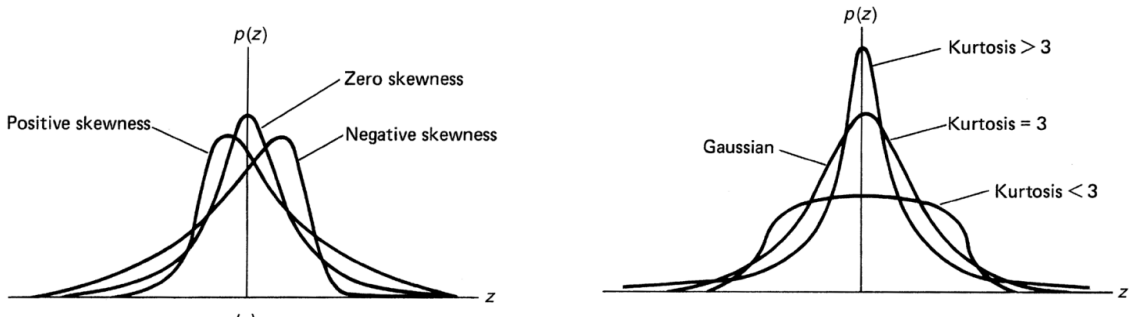


Figure 3.2: $Ku - Sk^2 - 1 \geq 0$

In Fig. 3.3 we can see how our probability distribution changes for varying Sk and Ku. As we can see a positive skewness bends our distribution to the rights, whereas a negative distribution bends it to the left. If the kurtosis is $Ku > 3$ our distribution sharpens, whereas for $Ku < 3$ it flattens. In terms of surfaces, we can see in Fig. 3.4 how the shape changes for varying values of skewness and kurtosis (the idea is similar).



(a) Skewness

(b) Kurtosis

Figure 3.3: How the shape of a probability distribution changes for varying Sk and Ku.

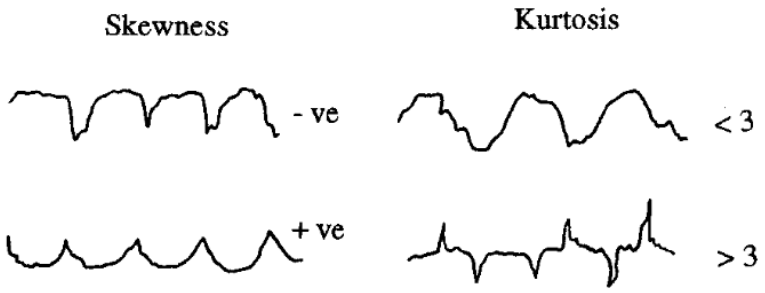


Figure 3.4: How the shape of a surface changes for varying Sk and Ku.

Assuming that we have a rough surface in three dimensions with height function $z(n, m)$, then to find the statistical moments we use the relation,

$$u_v = \mathbb{E}\left[(z - \bar{z})^v\right] = \sum_{k=0}^{m-1}\sum_{l=0}^{n-1}\frac{(z_{k,l} - \bar{z})^v}{mn} \tag{3.4}$$

And thus, we can write the first four statistical moments as,

$$\mu = \sum_{k=0}^{m-1}\sum_{l=0}^{n-1}\frac{z_{k,l}}{mn} \tag{3.5}$$

$$\sigma = \sqrt{\sum_{k=0}^{m-1}\sum_{l=0}^{n-1}\frac{(z_{k,l} - \mu)^2}{mn}} \tag{3.6}$$

$$Sk = \sum_{k=0}^{m-1}\sum_{l=0}^{n-1}\frac{(z_{k,l} - \mu)^3}{mn\sigma^3} \tag{3.7}$$

$$Ku = \sum_{k=0}^{m-1}\sum_{l=0}^{n-1}\frac{(z_{k,l} - \mu)^4}{mn\sigma^4} \tag{3.8}$$

These are some of the most fundamental parameters to characterize a surface, and therefore they must be in the input of a rough surfaces simulation algorithm. However, these parameters are not the only parameters that can characterize a surface.

3.2.2 Amplitude Parameters

These parameters give the information about the vertical changes in respect to a level of reference. Two of the most important statistical quantities that describe the amplitude are: the arithmetic mean R_a and the standard deviation R_q. The main difference between the previous statistical measures is that now we compute the mean of the absolute value of the height function $y(x)$ and we compute it in respect to one specific dimension (not two). The mathematical relations that describe these quantities for the continuous and discrete spectrum are,

$$R_a = \frac{1}{l}\int_0^l |y(x)|\,dx \qquad R_a = \frac{1}{n}\sum_{i=1}^{n}|y_i| \tag{3.9}$$

$$R_q = \sqrt{\frac{1}{l}\int_0^l y^2(x)dx}, \qquad R_q = \sqrt{\frac{1}{n}\sum_{i=1}^{n}y_i^2} \tag{3.10}$$

In these manner, we can also define the skewness and the kurtosis,

$$R_{sk} = \frac{1}{R_q^3}\sqrt{\int_{-\infty}^{+\infty} y^3(x)dx}, \qquad R_{sk} = \frac{1}{nR_q^3}\sum_{i=1}^{n} y_i^3 \tag{3.11}$$

$$R_{ku} = \frac{1}{R_q^4}\sqrt{\int_{-\infty}^{+\infty} y^4(x)dx}, \qquad R_{ku} = \frac{1}{nR_q^4}\sum_{i=1}^{n} y_i^4 \tag{3.12}$$

3.2.3 Spatial Parameters

The statistical parameters of the spatial characteristics are mainly determined from *ACF* or *PSD*. For a continuous normalized height function $y(x)$, we can define correlation function *ACF* as,

$$ACF(r_x) = \frac{1}{R_q^2(l - r_x)}\int_0^{l-r_x} y(x)y(x + r_x)dx \tag{3.13}$$

where l is the profile length in x-axis direction, and r_x is the distance between two profile points. In the discrete case,

$$ACF(r_x) = \frac{1}{R_q^2(l - r_x)}\sum_{x=1}^{l-r_x} (y(x) - < y >)\,(y(x + r_x) - < y >) \tag{3.14}$$

Many surfaces or profiles with small r_x can represented from *ACF* of the exponential form,

$$ACF(r_x) = \exp(-r_x/\beta) \tag{3.15}$$

The measure that characterizes how fast this function decreases, is called *correlation length*. For a particular surface or profile, correlation length is the length for which the correlation function decreases in a particular percentage (usually the 10% of the initial value).

Another important function that it is used for the spatial characterization of surfaces is PSD. It includes the frequency interval from zero (flat surface) to infinity (infinite continuous peaks and valleys). It is defined as the Fast Fourier Transform of ACF,

$$PSD(f_r) = \left|\sum_{k=0}^{m-1} ACF(k)\exp\left(-\frac{2\pi kr}{m}\right)\right| \tag{3.16}$$

where $r = 0, 1, \ldots, m-1$ and f_x is the frequency in x-axis with $f_x = r/(m\Delta x)$.

3.3 Gaussian Surfaces Simulation

3.3.1 The recipe

To simulate Gaussian surfaces we must follows the steps that are described in the paper [15]. The whole method is basically based in Fourier transforms, and even if all these steps look a bit scary, there are ready python libraries that can do all the computations without a lot of effort.

1. The discrete form of ACF R(m, n) is extracted from the given ACF $f(x, y)$.

- The input ACF should be defined at first. For example, the following exponential form of ACF $f(x, y)$ is widely quoted.

$$f(x, y) = \sigma^2 \exp\left[-2.3\sqrt{\left(\frac{x'}{\beta_x}\right)^2 + \left(\frac{y'}{\beta_y}\right)^2}\right] \tag{3.17}$$

where

$$x' = x\cos\phi + y\sin\phi, y' = -x\sin\phi + y\cos\phi \tag{3.18}$$

where σ is the standard deviation of height sequence; β_x and β_y stand for the auto-correlation lengths in x and y directions, respectively; ϕ stands for the prescriptive orientation of surface texture.

- An equal discrete spacing in x and y directions needs to be specified (for example, $\Delta x = \Delta y = 1\ \mu m$) and then, discretize the given ACF into sequence $R(m+1, n+1)$ within the range of $-m/2 \le x \le m/2$ and $-n/2 \le y \le n/2$.

$$R\left(g + \frac{m}{2} + 1, h + \frac{n}{2} + 1\right) = f(g, h) \tag{3.19}$$

where m and n are numbers of surface points in x and y directions; $g = -m/2, -m/2+1, \ldots, m/2$; $h = -n/2, -n/2+1, \ldots, n/2$.

- One of the repeated rows $m/2+1$ or $m/2+2$, and the repeated columns $n/2+1$ or $n/2+2$ in the sequence $R(m+1, n+1)$ is required to be deleted to generate the sequence $R(m, n)$ of the same size with the target rough surface.

2. The PSD and TF can be obtained from $R(m, n)$,

- FFT method is applied here to get the PSD $P(I, J)$.
- Since the PSD of white noise is a constant C, assuming $C = 1$, the transfer function $H(m, n)$ can be obtained:

$$H = \sqrt{\frac{P}{C}} = \sqrt{P} \tag{3.20}$$

3. The phase sequence $\Phi(m, n)$ can be produced from the white noise sequence $\eta(m, n)$ which needs to be updated with inverse fast Fourier transform (IFFT) method.

- The white noise generator `randn(m, n)` is used to obtain the white noise sequence $\eta(m, n)$.
- The phase sequence $\Phi(m, n)$ is generated from the sequence $\eta(m, n)$ with

$$\Phi(k+1, l+1) = 2\tan^{-1}\left(\frac{-\sum_{r=0}^{m-1}\sum_{s=0}^{n-1}\eta(r+1, s+1)\sin\left(\frac{2\pi kr}{m} + \frac{2\pi ls}{n}\right)}{\sum_{r=0}^{m-1}\sum_{s=0}^{n-1}\eta(r+1, s+1)\cos\left(\frac{2\pi kr}{m} + \frac{2\pi ls}{n}\right)}\right) \tag{3.21}$$

where all the elements in the phase sequence Φ range from 0 to 2π, and the computation of $\Phi(m, n)$ can be conducted with FFT method to improve the efficiency.

- The white noise sequence $\eta(m,n)$ is updated from $\Phi(m,n)$ with IFFT method,

$$\eta(k+1,l+1) = \sum_{r=0}^{m-1}\sum_{s=0}^{n-1} \exp\left(i\Phi + \frac{2\pi ikr}{m} + \frac{2\pi ils}{n}\right) \tag{3.22}$$

- FFT method is then introduced to the updated the white noise sequence $\eta(m,n)$,

$$A(k+1,l+1) = \sum_{r=0}^{m-1}\sum_{s=0}^{n-1} \eta(r+1,s+1)e^{-2\pi i(kr/m+ls/n)} \approx nm \cdot e^{i\Phi} \tag{3.23}$$

4. The Gaussian sequence $z_0(m,n)$ is generated from the dot-product of A and H with IFFT method.

$$z_0(k+1,l+1) = \frac{1}{mn}\sum_{r=0}^{m-1}\sum_{s=0}^{n-1}(A\cdot H)e^{2\pi i(kr/m+ls/n)} \tag{3.24}$$

5. Repeat steps 3 $\sim$ 4, if the skewness and kurtosis values of z_0 approach 0 and 3, respectively, otherwise, repeat the step 2.

6. The Gaussian sequence z_0 is required to be scaled and translated to obtain the target Gaussian height sequence.

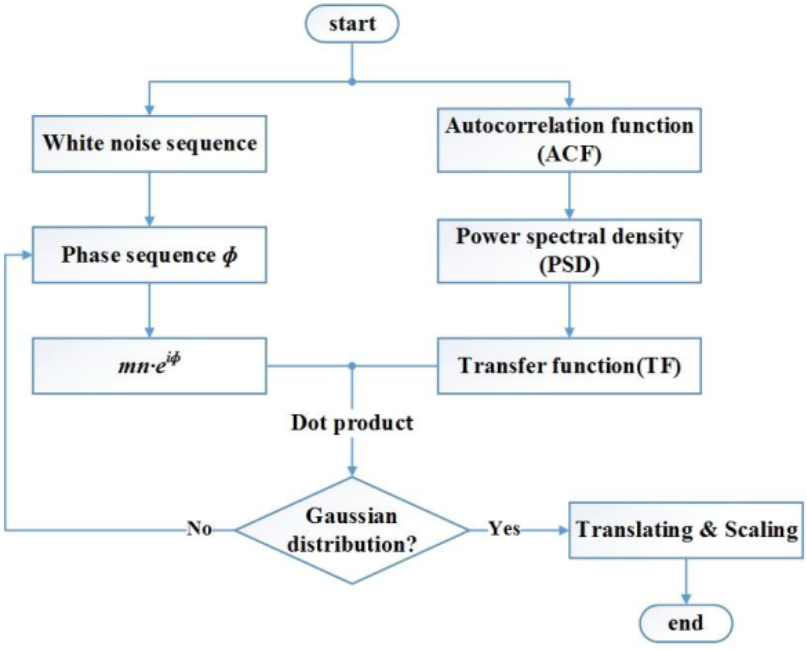


Figure 3.5: A flow chart that represents the algorithm with which we produce Gaussian Surfaces.

Computational Note: This algorithm may seem scary to the eyes of a new researcher who wants to apply this method, but fortunately we have created the Git-Hub repository for you. All you must do is to visit the Git-Hub url `https://github.com/BlackPianoCat/simulating_non_gaussian_surfaces` and then you will have a python rough surfaces simulator. Attention: even if we succeeded to translate Gaussian surface simulation from MATLAB to python, we failed to translate Non-Gaussian simulator, because there is a MATLAB function that doesn't exist in python. We will explain the reason soon, however if you want to use our code, take care.

3.3.2 The relation between Gaussian Surfaces and Ising model

At first sight, this is a completely different algorithm from this one of Metropolis-Hastings. Gaussian surfaces simulation is a simulation which is based in a completely geometrical perception, whereas behind Metropolis-Hastings algorithm lurks a dynamical model which can interpret the physical phenomenon of ferromagnetism.

If we succeed to do all these steps that we described in the previous subsection, we end up in a function F_{Gauss} which has as its input parameters the number of points in each axis N, standard deviation σ, correlation lengths ξ_x and ξ_y, and the fractality a. The output is a Gaussian rough surface. Therefore we can write,

$$F_{\text{Gauss}}(N, \sigma, \xi_x, \xi_y, a). \tag{3.25}$$

In this surface skewness and kurtosis parameters are always the same $Sk = 0$ and $Ku = 3$.

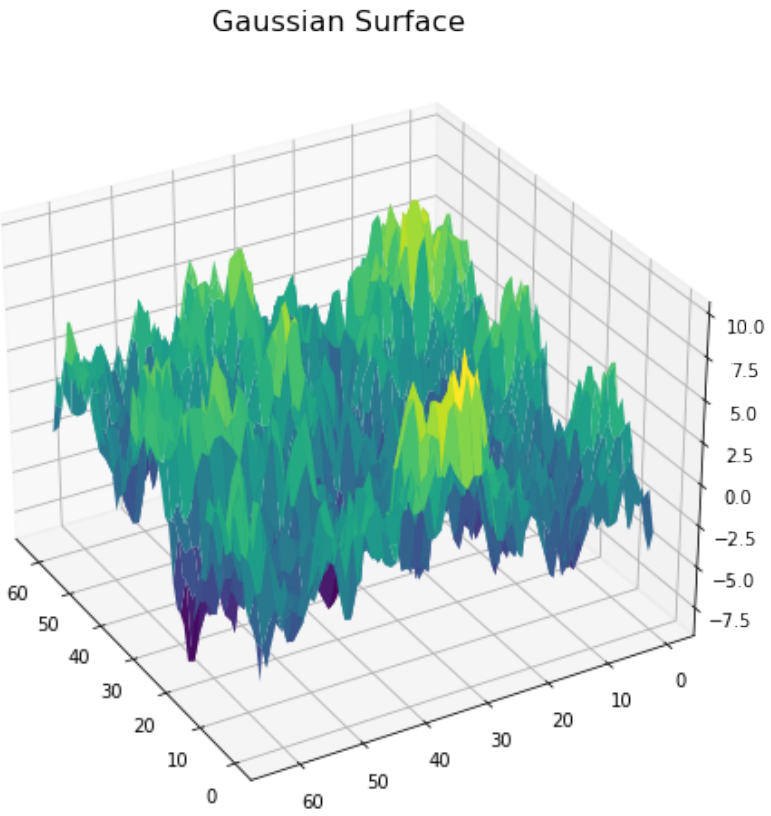
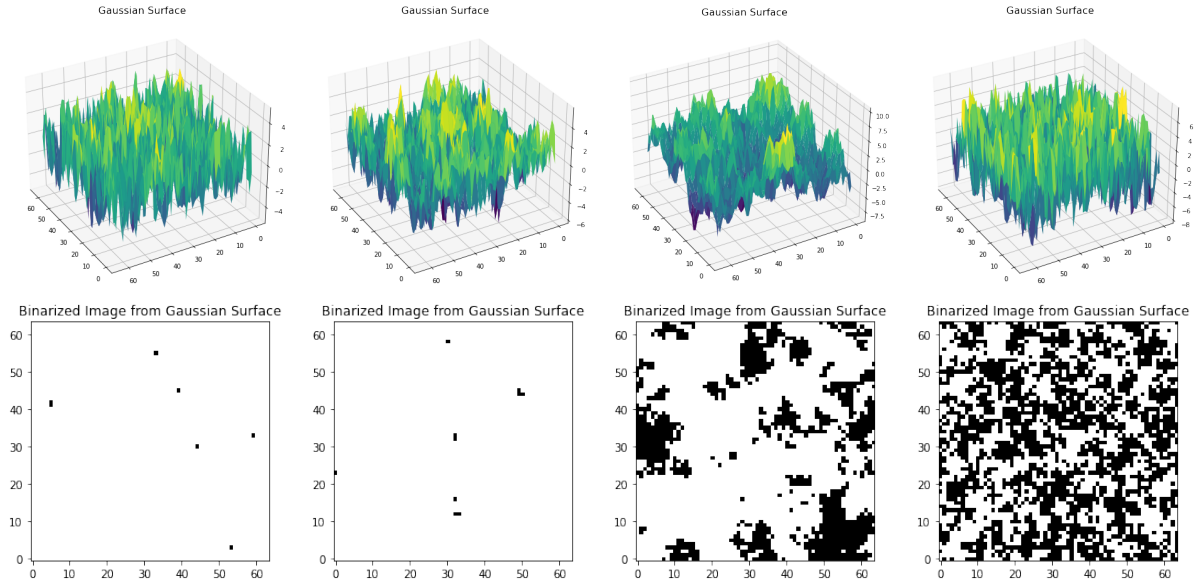


Figure 3.6: An example of a Gaussian surface with $N = 64$, $\sigma = 3$, $\xi_x = \xi_y = 3.8$ and $a = 0.6$.

This function can produce some cool plots like this in Fig. 3.6, but these are three dimensional plots, whereas Ising model we work in this thesis is two dimensional. However, there is a very simple way to translate a three dimensional surface to a two dimensional image, and this is by choosing a specific threshold. Imagine that there is a plane parallel to x-y plane that intersects our surface. The regions that are over the surface are white pixels, and the regions under the surface are black pixels. In this manner we put another one parameter in our surface: the parameter of threshold t, and thus we can write our function as,

$$F_{\text{Gauss}}(N, \sigma, \xi_x, \xi_y, a, t) \tag{3.26}$$

Having done that , we can produce images like these in Fig. 3.7. These images look pretty similar to Ising lattices. For instance, the first two remind us Ising model in low temperatures, the third one remind us the behavior near the critical point and the last one high temperatures.



(a) $\xi = 1, t = 4.5$ (b) $\xi = 2.5,\ t = 5$ (c) $\xi = 3.8,\ t = 2.0$ (d) $\xi = 1.5,\ t = 0$

Figure 3.7: The lattices that are produced from different input parameters of Gaussian surfaces simulation ($N = 64,\ \sigma = 3,\ \xi_x = \xi_y = \xi,\ a = 0.6$).

3.3.3 The distribution of magnetization

The first question that comes to our mind is about magnetization. If there is one measure that can describe both Gaussian Surfaces and Metropolis-Hastings simulation, is this. However, we saw that the magnetization has some fluctuations over time that become more intense near the critical point. The problem with Gaussian Surfaces simulation is that there is no time on it. The only way to think about time in Gaussian simulation is by repeating the same simulation for the same parameters and see how magnetization fluctuates over time.

We have done this process in Fig. 3.8. As we can see the time series of magnetization is Gaussian distributed and does small fluctuations near its mean value. In Fig.3.9 we can see the probability distribution of the magnetization, and it is clear that for bigger correlation lengths we have more intense fluctuations of magnetization. However,

in rough surfaces simulation, the probability distribution is always Gaussian, whereas in Metropolis-Hastings the probability distribution flattens at the critical region. For both figures we made the assumption $N = 64$, $\sigma = 3$, $\xi_x = \xi_y = \xi$, $a = 0.6$.

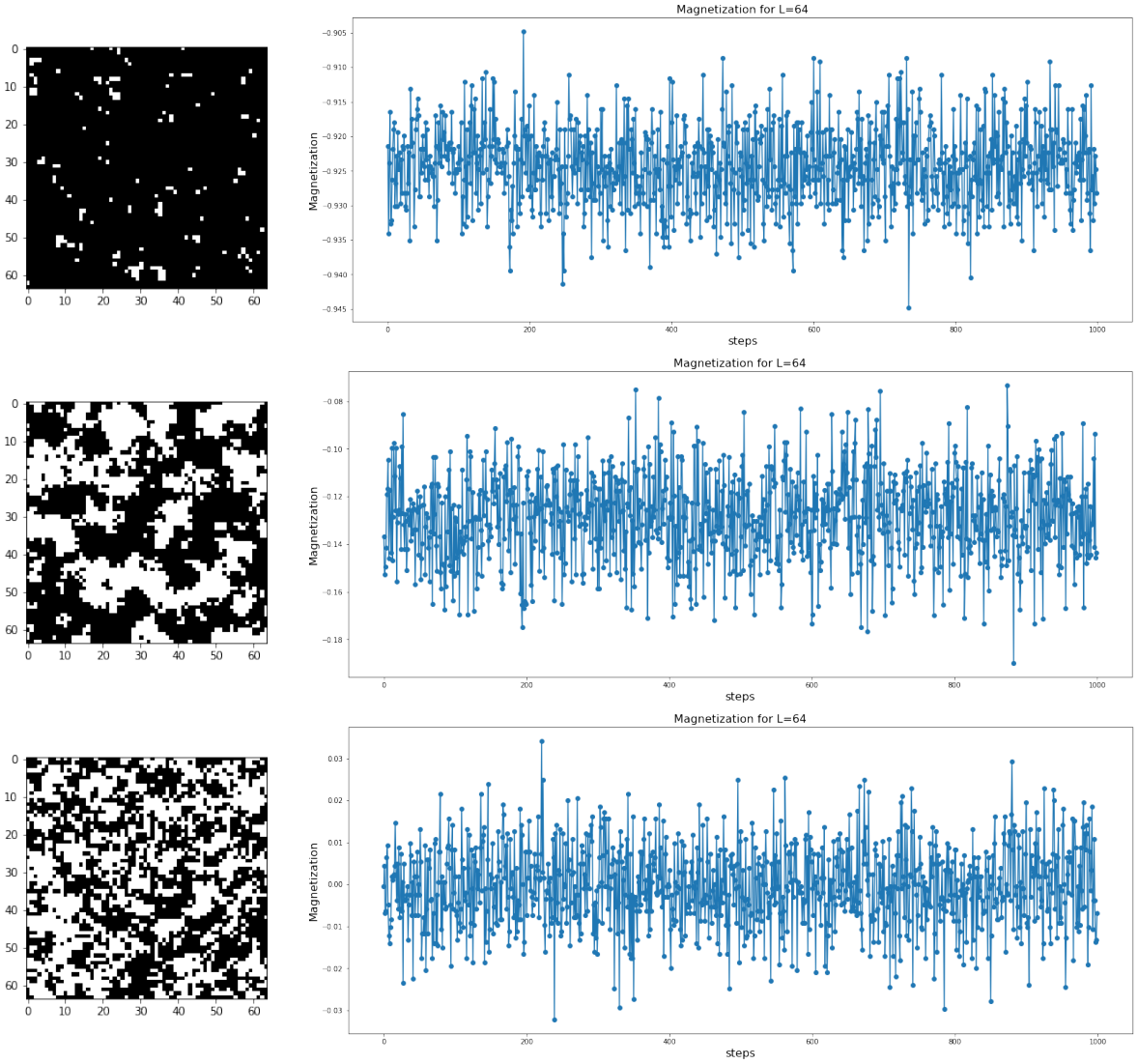
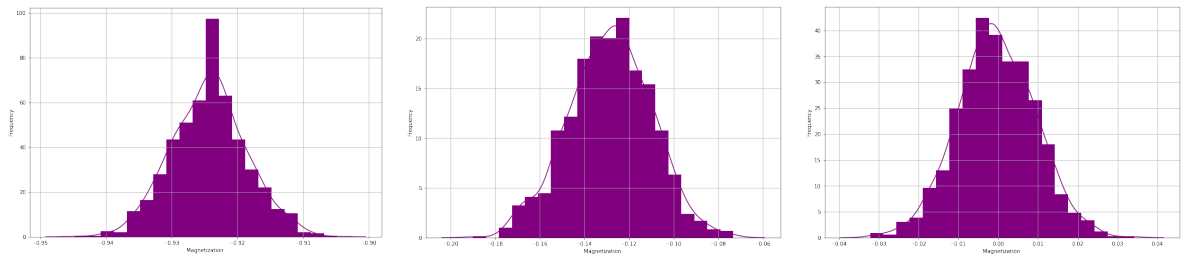


Figure 3.8: Time series of magnetization: in the first row we have $\xi_x = \xi_y = 1$ and $t = 5$, in the second one $\xi_x = \xi_y = 4$ and $t = 0.5$, and in the third row $\xi_x = \xi_y = 1.5$ and $t = 0$.

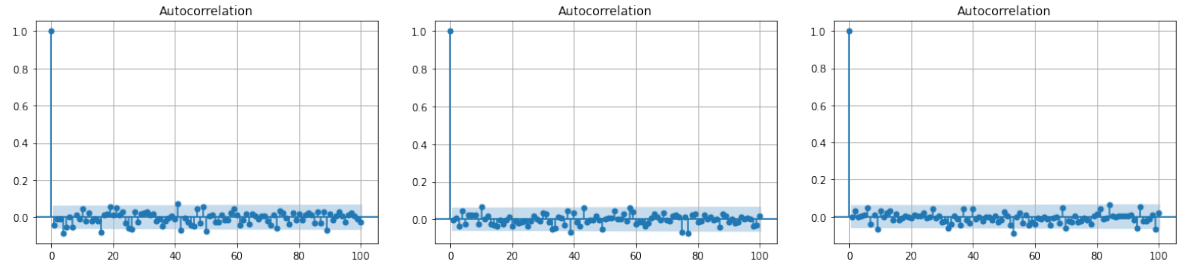


(a) $\xi = 1$, $t = 5$ (b) $\xi = 4$, $t = 0.5$ (c) $\xi = 1.5$, $t = 0$

Figure 3.9: The probability distribution of magnetization for different input parameters of Gaussian surfaces simulation.

3.3.4 The autocorrelations of magnetization

So in this chapter we are trying to answer the question: *why a mapping between Gaussian surfaces and Ising model may be useful?* The first answer we can give is that we can reach the equilibrium instantly without doing thousands of steps. However, there is another very important reason to do something like that: *we don't need a sampling rate*. This is because the mechanism we produce Gaussian surfaces doesn't use any Markov chain (or stochastic process generally) that changes each spin separately. Nevertheless, each experiment for some fixed parameters ξ, t is completely independent from the previous one, and therefore we can produce uncorrelated samples very fast and easily.



(a) $\xi = 1,\ t = 5$ (b) $\xi = 4,\ t = 0.5$ (c) $\xi = 1.5,\ t = 0$

Figure 3.10: Autocorrelations of magnetization for three different options of parameters. We see that the autocorrelations of magnetization are very small, even if we simulate lattices that look similar to those of Ising model in critical region (b).

3.4 Non-Gaussian Surfaces Simulation

In this section we discuss about the way we can translate a Gaussian surface to a non-Gaussian. However, in this thesis we mainly focused on the Gaussian surfaces simulation and this is because of two reasons: (1) they are simpler and we can understand better what we do, (2) we had some convergence problems with the non-Gaussian translator system. The main reason why we had this problem was that we started our work in `python`, so we had to translate the MATLAB algorithm to python and some functions didn't exist or they didn't work in the way we wanted.

3.4.1 Non-Gaussian Translator System

There are two non-Gaussian translator systems in bibliography: Johnson's and Pearson's. Both of them can generate height sequences and can cover the whole $Ku - Sk^2 - 1 \geq 0$ plane. Concerning Johnson's system, there are three types of fitting methods: S_B, S_U and S_L.
Therefore we have,

1. The lognormal distribution S_L,

$$z_2 = \xi + \lambda e^{(\eta-\gamma)/\delta}, \quad (\xi < \eta) \tag{3.27}$$

2. The bounded distribution S_B,

$$z_2 = \xi + \frac{\lambda e^{(\eta-\gamma)/\delta}}{1 + e^{(\eta-\gamma)/\delta}}, \quad (\xi < \eta < \xi + \lambda) \tag{3.28}$$

3. The unbounded distribution S_U,

$$z_2 = \xi + \lambda \sinh\left(\frac{\eta - \gamma}{\delta}\right) \tag{3.29}$$

where η is the white noise sequence, z_2 is the non-Gaussian sequence, γ and δ are local and shape parameters, and λ and ξ are proportional coefficient and position parameter, respectively. The value of all the parameters γ, δ, λ and ξ can be determined by skewness and kurtosis. Nevertheless, even if in MATLAB is very easy to do that, in python there is not any function that can find these parameters from skewness and kurtosis. There is only one Git-Hub repository from Max Pierini `https://github.com/maxdevblock/j_johnson_M` for Johnson S_U translator system.

The other translation system is Pearson's which uses a probability distribution that satisfy the following condition,

$$\frac{d(\log(p(x)))}{dx} = -\frac{a + x}{c_0 + c_1 x + c_2 x^2} \tag{3.30}$$

where a, c_0 , c_1 and c_2 are constants that can be obtained with given standard deviation, skewness, kurtosis and the probability function derived from the above differential equation.

3.4.2 Non-Gaussian Transformation

The complicated filter process is eliminated and height sequences with any input Sk and Ku in the whole skewness-kurtosis plane ($Ku - Sk - 1 \geq 0$) can be obtained by carrying out the following steps.

1. The white noise generator `randn(m,n)` is utilized to obtain the white noise sequence $\eta(m, n)$ and transform it to a non-Gaussian sequence $z_2(m, n)$ with Johnsons translator system in terms of the input first four moments; If S_B or S_U are unable to converge, use the Pearsons translator system to obtain the non-Gaussian sequence $z_2(m, n)$, instead.

2. The skewness and kurtosis of the sequence $z_2(m, n)$ is calculated and repeat the calculation if it doesn't converge.

3. The sequence $z_2(m, n)$ needs to be scaled and translated with $z_3 = \sigma \cdot z_2/\sigma_1 - \mu_1 + \mu$ (σ_1 and 1 stand for the standard deviation and the mean of the sequence z_2) to update the non-Gaussian sequence to satisfy the given first four moments.

3.4.3 Restructure and rearrangement of the heigh sequences

The restructure and rearrangement methods of height sequences are utilized to make the target non-Gaussian height sequence satisfy the given ACFs in x and y directions on the basis of the spatial distribution of the corresponding Gaussian rough surfaces.

1. The simulated Gaussian sequence $z_G(m, n)$ and the non-Gaussian sequence $z_3(m, n)$ need to be restructured into sequences $q(m \cdot n, 1)$ and $Q(m \cdot n, 1)$, respectively.

2. The sorting and indexing tools in Matlab are introduced to arrange the sequence $q(m \cdot n, 1)$ and $Q(m \cdot n, 1)$ then, rearrange the sequence $Q(m \cdot n, 1)$ in terms of the sorting index of the sequence $q(m \cdot n, 1)$.

3. The rearranged sequence $Q(m \cdot n, 1)$ is required to get restructured into the target non-Gaussian height sequence $z_N(m, n)$ which has almost the same ACFs with those of the corresponding Gaussian height sequence z_G.

3.4.4 An example of Non-Gaussian Surface

We can produce a non-Gaussian surface using python, however the results are not the best with our algorithm. We used Johnson S_U translator system. As we can see in Fig. 3.11, there are some points on the surface that don't converge. Normally, we don't expect to have so high values in z-axis. Of course we can still produce two dimensional lattices, and we still can try to create Machine Learning algorithms even with these surfaces. Nevertheless, if the simulation doesn't converge, we lose the physical meaning, and thus is better to work with Gaussian Surfaces.

Moreover, there is another reason why we chose to work only with Gaussian surfaces. In the previous subsection, we saw that we can produce two-dimensional images that look very similar to Ising lattices only by changing correlation length ξ_x, ξ_y and threshold t. Apart from that, we saw in the second chapter that all the physics of Ising model is hidden in first and second order statistics. Therefore, an investigation of Gaussian surfaces is enough for this kind of research.

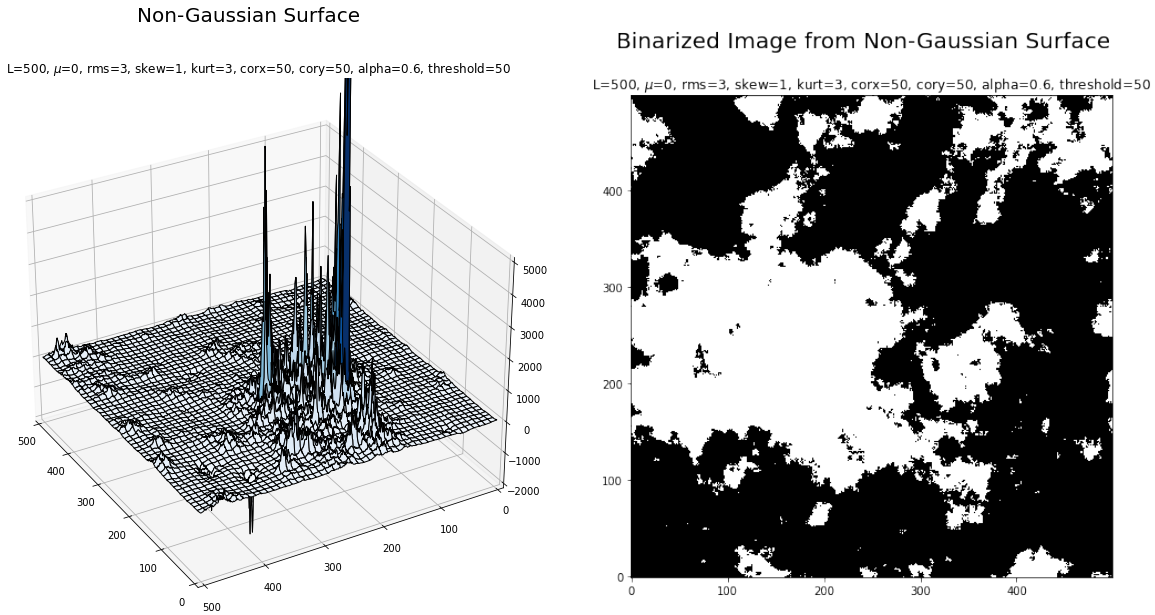


Figure 3.11: Non-Gaussian surface and lattice ($N = 500$, $\mu = 0$, $\sigma = 3$, $Sk = 1$, $Ku = 3$, $\xi_x = \xi_y = 50$, $a = 0.6$, $t = 50$).

Part II

Main Part of Master's Thesis

Phenomena complex - laws simple... Know what to leave out.

- Richard P. Feynman

4 Complexity Measures

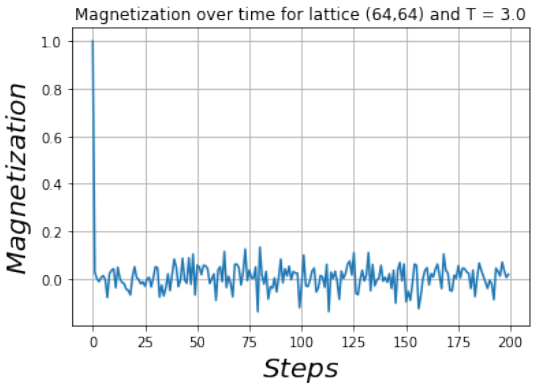
In this chapter we will define some complexity measures for Ising model. By the term 'complexity measure', we mean a measure that somehow can characterize the structure of our lattice. Ideally, we expect to have a maximum or minimum in the critical temperature because in this temperature the equilibrium configuration lattice becomes fractal [26]. Initially, we present some results from Metropolis-Hastings algorithm, since it is important to understand the meaning of statistical quantities we compute in Ising model and have a clear view about the diagnostics of our algorithm. By the term 'diagnostics', we mean some techniques that help us to understand if the algorithm runs correctly and if our system has reached the equilibrium. Having done that, we will compute some complexity measures from bibliography, and then we will define some new ones.

4.1 Running Metropolis-Hastings algorithm

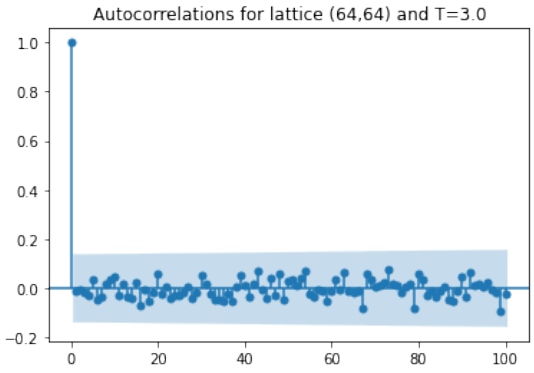
Even if we have described the algorithm that can help us to simulate the Boltzmann distribution, there are still problems. The two main aspects of this statement are related to the time we need to reach the equilibrium and the way we do sampling to compute the quantities of interest correctly.

The best way to know if we have reached the equilibrium is to look *the behavior of ergodic mean over time*. In our case, the mean magnetization plays the role of ergodic mean. Therefore, what we do is to plot mean magnetization over time and see if it continues to fluctuate or if it has reached a stationary state. However, the fluctuations of the mean magnetization depend on temperature. Specifically, if we sample near the critical point, the fluctuations become very intense. So we have to take attention. Additionally, there is another one thing we must take care of, and this is the *sampling frequency f* (or *thinning* in statistics). It is very important not to take sample in each step from our chain because of theoretical and computational reasons. The theoretical reason is that, if we want to do sampling using the ergodic theorem, we need to have independent identically distributed random variables. This means that there must not be autocorrelations in our sample. Therefore, the only way to reach our goal is by sampling with a frequency f and discard all the other samples.

The other reason is that if we do sampling in every step, we may have problems with our memory.



(a) Mean magnetization over time.

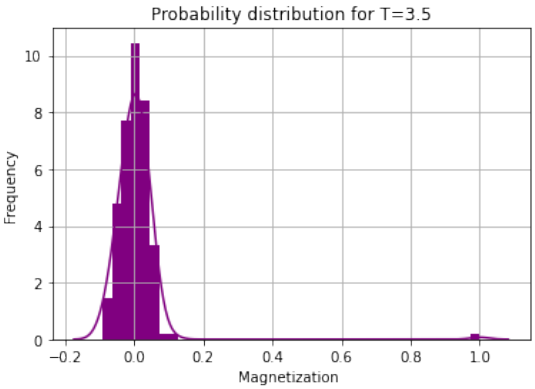


(b) Autocorrelations.

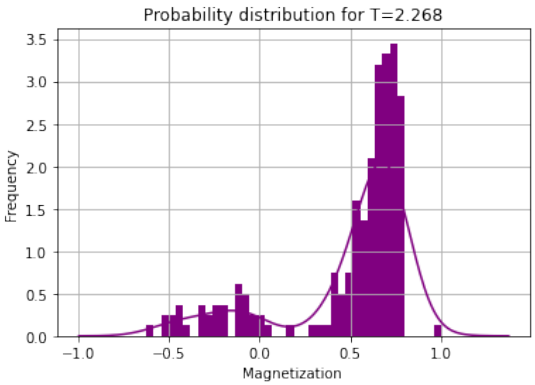
Figure 4.1: The diagnostics of a healthy Metropolis-Hastings run.

In Fig. 4.1 we have run Metropolis-Hastings algorithm for 10^8 steps and sampling frequency 5×10^5. The temperature of this run is $T = 3$ and the lattice is $(64, 64)$. As we can see the mean magnetization reaches its equilibrium very fast, and there are no statistically significant autocorrelations. Even if the system has reached its equilibrium very fast due to the fact that we chose the appropriate number of steps and frequency, this is not always the case. The period before our system reaches its equilibrium is called *burn-in period*, and we must discard all the samples of this time interval.

Another very important feature of a run is the probability distribution of the mean magnetization, and the fluctuations of its' time series. As we said before, the fluctuations become more and more intense as we approach the critical temperature. In Fig. 4.2 we can see the probability distributions of the mean magnetization for a system in temperature $T = 3$ and another one in temperature $T = 2.268$, which is inside the critical region. It is clear that near the critical region, the things go insane.



(a) $T = 3$

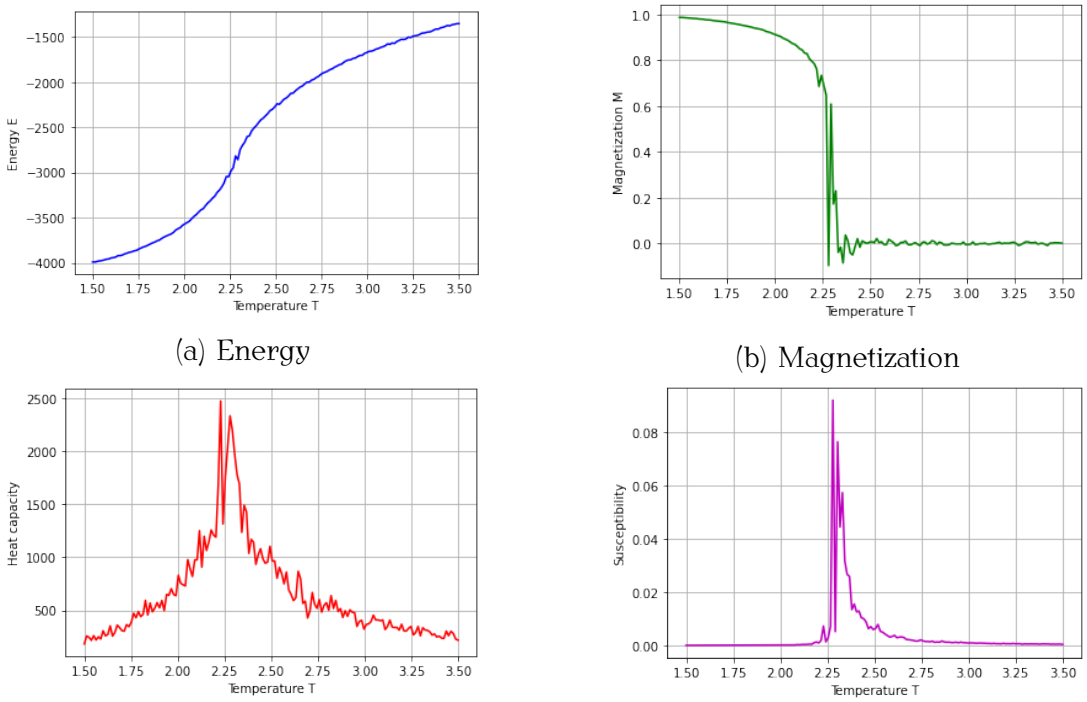


(b) $T = 2.268$

Figure 4.2: Probability distribution of Mean Magnetization.

In Fig. 4.3 we have run Metropolis-Hastings algorithm for 160 different temperatures in interval $T \in [1.5, 4.0]$. In each run we do some steps in equilibrium with frequency $f = 5 \times 10^5$. We compute the energy and the magnetization in each of

these steps and then we compute their averages for each temperature. These plots are in the Fig. 4.3a and 4.3b. Let's see for a bit the shape of the curve of magnetization. We have a very big magnetization in low temperatures and approximately zero magnetization in high temperatures. The fluctuations of mean magnetization are indicated from the heat capacity (Fig. 4.3c), which is the variance of energy over time and susceptibility in Fig. 4.3d, which is the variance of magnetization over time. What these figures say to us, is that when our system is in critical region, the attitude of fluctuations is maximum.



(a) Energy

(b) Magnetization

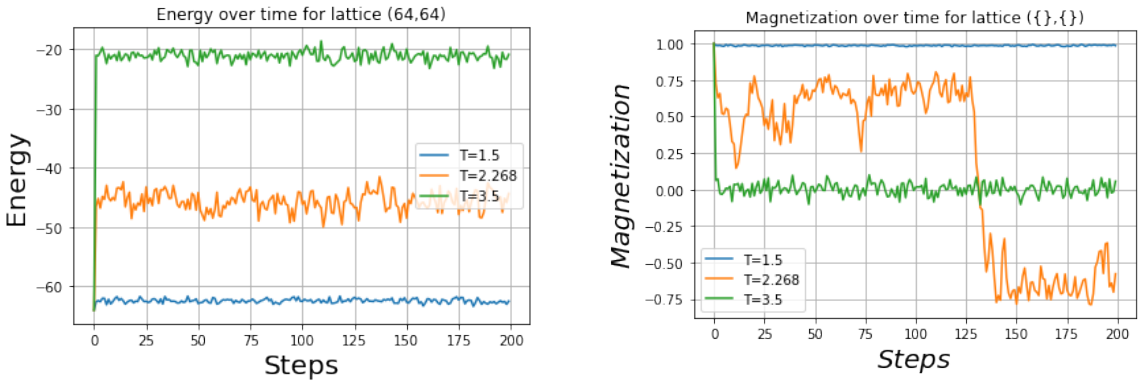
(c) Heat Capacity

(d) Susceptibility

Figure 4.3: Some of the first and second order statistics of Ising Model.

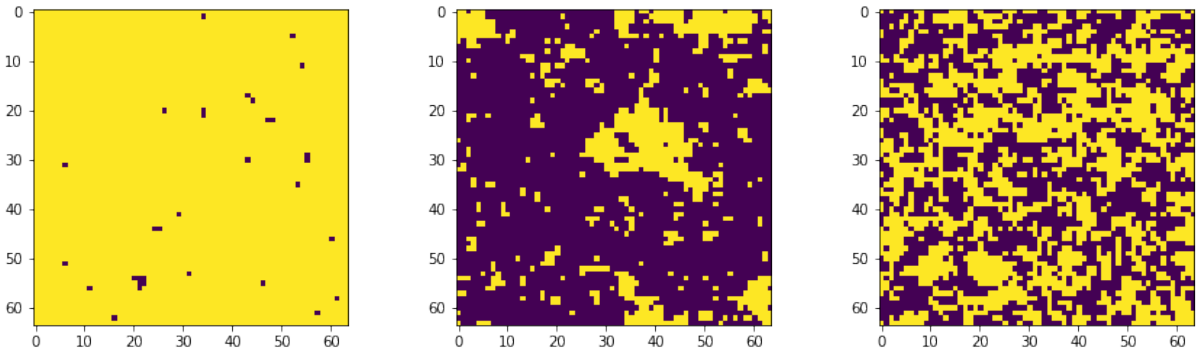
In Fig. 4.4 we can see the fluctuations of energy and magnetization in their time series. We have run three Metropolis-Hastings algorithms for three different temperatures with the same steps and sampling frequency as before. Perhaps Fig. 4.4b is the most clear picture of these fluctuations.

For this thesis it is also very essential to have a geometrical intuition for what happens in our lattice as we change the temperature of our system. In Fig. 4.5 we can see three configurations of Ising Model for three different temperatures, having reached the equilibrium. In high temperatures (c) there is symmetry. The mean magnetization is zero, because the number of positive spins is approximately equal to the number of negative spins. As the system approaches the critical temperature (b), the symmetry breaks. Therefore, in critical point aggregates are shaped and we have a very high fractality. For low temperatures (a) there is a dominant state (up or down). We only have some small dots of different spins. However, if we had to choose which of these three configurations is the most assymetric or complex, we would choose the second one which is in the critical temperature. And this is something very important, because as we will see in the next chapter, this property makes this configuration the most complex one.



(a) Energy (b) Magnetization

Figure 4.4: The times series of energy and magnetization for three different temperatures.



(a) $T = 1.5$ (b) $T = 2.268$ (c) $T = 3.8$

Figure 4.5: The equilibrium configurations of our system for three different temperatures.

A note about the computational time: Python is a high level programming language with a lot of tools and functions, however it is not the faster one. Metropolis-Hastings algorithm it is a very time demanding algorithm and this can cause a very big problem. Fortunately, there is a python library called `Numba` (`https://numba.pydata.org/`), which can accelerate scripts written in `numpy`. A very nice example of Metropolis-Hastings algorithm with `Numba` can be found in [38]. Furthermore, because even if we use `Numba`, the computations are still time consuming, we propose platforms like Kaggle that allow people to run their codes online with a maximum time restriction of nine hours[1].

4.2 Classification Complexity Measures

In this section, we see Ising model configurations (like these in Fig. 4.5) as two dimensional classification problems, where the spins are the labels and coordinates are the features. This is, of course, a geometrical approach of our problem. Our start point for this research was the paper of Ana C. Lorena et al [10], in which the authors

[1]This time restriction changes over time. We just mention an indicative time restriction, which was valid at the moment that this thesis was written.

propose a collection of measures that characterize the complexity of a classification problem.

The authors group these measures as follows:

- **Feature-based measures**, which characterize how informative the available features are to separate the classes
- **Linearity measures**, which try to quantify whether the classes can be linearly separated;
- **Neighborhood measures**, which characterize the presence and density of same or different classes in local neighborhoods;
- **Network measures**, which extract structural information from the dataset by modeling it as a graph;
- **Dimensionality measures**, which evaluate data sparsity based on the number of samples relative to the data dimensionality;
- **Class imbalance measures**, which consider the ratio of the numbers of examples between classes.

In this thesis we didn't compute all of these measures, but we mainly focused on some feature-based, linearity, neighborhood and class-imbalance measures, since we saw that the most of them are unable to identify the criticality (apart from one).

To define the measures, we consider that they are estimated from a learning dataset T (or part of it) containing n pairs of examples (x_i, y_i), where $x_i = (x_{i1}, \ldots, x_{im})$ and $y_i \in \{1, \ldots, n_c\}$. That is, each example x_i is described by m predictive features and has a label y_i out of n_c classes.

4.2.1 Feature-based measures

These measures evaluate the discriminative power of our features. If there is at least one very discriminative feature in the dataset, the problem can be considered simpler than if there is no such attribute.

Maximum Fisher's Discriminant Ratio (F1)

This ratio measures the overlap between the values of the features in different classes, and it is given by

$$F1 = \frac{1}{1 + \max_{i=1}^{m} r_{fi}} \tag{4.1}$$

where,

$$r_{fi} = \frac{\sum_{j=1}^{n_c} \sum_{k=1, k\neq j}^{n_c} p_{c_j} p_{c_k} \left(\mu_{c_j}^{f_i} - \mu_{c_k}^{f_i}\right)^2}{\sum_{j=1}^{n_c} p_{c_j} (\sigma_{c_j}^{f_i})^2} \tag{4.2}$$

where f_i stands for the feature i, n_{c_j} is the number of examples in the class c_j, p_{c_j} is the proportion of examples in the class c_j, $\mu_{c_j}^{f_i}$ is the mean of feature f_i across examples of class c_j, and $\sigma_{c_j}^{f_i}$ is the standard deviation of such values. To understand what this measure actually does, let's take the example of the Fig. 4.6. Here f_1 is the most discriminative feature, whereas f_2 is absolutely non-discriminative. $F1$ measure

considers the means of the most discriminative feature, which in this case is f_1. In this manner, higher values of $F1$ indicate more complex problems. Note that $F1$ is bounded in the interval $(0, 1]$.

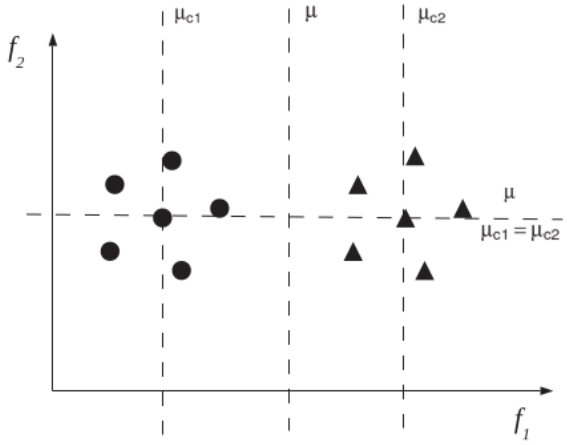


Figure 4.6: A diagram that help us understand how to compute $F1$ complexity measure.

Eq. 4.2 may seem a kind of complicated, but in case of Ising lattices, there are only two labels $s = 1$ and $s = -1$. Therefore, this equation takes the simpler form,

$$F1 = \frac{1}{1 + \max_{i=1}^{m} r_{f_i}}, \qquad r_{f_i} = \frac{p_{\{s=+1\}} p_{\{s=-1\}} (\mu^{f_i}_{\{s=+1\}} - \mu^{f_i}_{\{s=-1\}})^2}{p_{\{s=+1\}} (\sigma^{f_i}_{\{s=+1\}})^2 + p_{\{s=-1\}} (\sigma^{f_i}_{\{s=-1\}})^2} \tag{4.3}$$

where f_i are the x and y coordinates of Ising lattice. What we can do is to compute this $F1$ measure for one equilibrium configuration in each temperature of the Ising model. In Fig. 4.7 we have plot $F1$ against the temperature T. It is clear that there is a

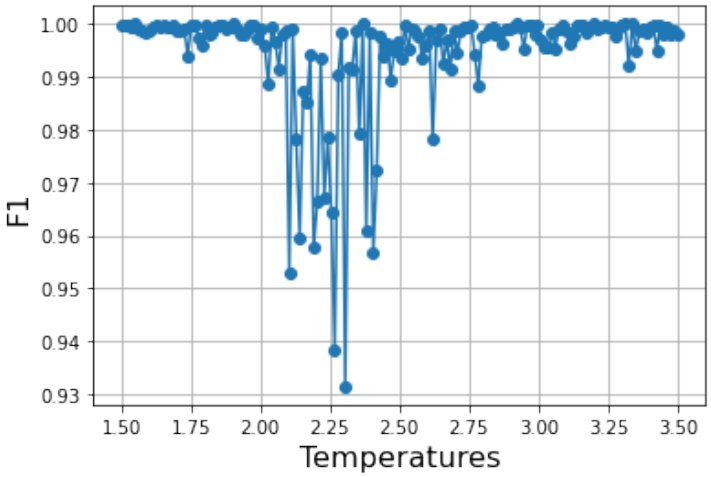


Figure 4.7: F1 measure for different temperatures (one configuration for each temperature). In this diagram we have 160 points on the interval $T \in [1.5, 3.5]$.

statistical global minimum, which means that we have the 'less complex' classification problem in the critical points in respect to this measure. However, we know that in

critical point our lattice becomes fractal, and we would expect that there would be a maximum in complexity. Let's say that *a 'less complex' classification problem, means that we have the 'most complex' Ising configuration*, and we will see the interpretation of this later.

Volume of Overlapping Region (F2)

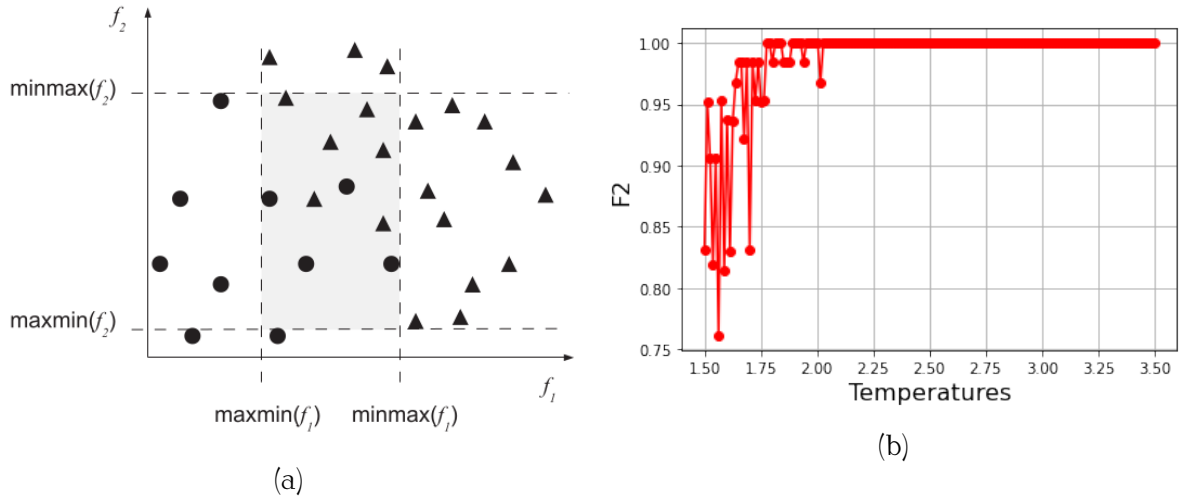
The F2 measure calculates the overlap of the distributions of the feature values within the classes. It can be determined by finding, for each feature f_i , its minimum and maximum values in the classes.

$$F2 = \prod_{i}^{m} \frac{\text{overlap}\ (f_i)}{\text{range}\,(f_i)} = \prod_{i}^{m} \frac{\max\{0, \min\max(f_i) - \max\min(f_i)\}}{\max\max(f_i) - \min\min(f_i)} \tag{4.4}$$

where

$$\begin{aligned} \min\max(f_i) &= \min\left(\max\left(f_i^{c1}\right), \max\left(f_i^{c2}\right)\right) \\ \max\min(f_i) &= \max\left(\min\left(f_i^{c1}\right), \min\left(f_i^{c2}\right)\right) \\ \max\max(f_i) &= \max\left(\max\left(f_i^{c1}\right), \max\left(f_i^{c2}\right)\right) \\ \min\min(f_i) &= \min\left(\min\left(f_i^{c1}\right), \min\left(f_i^{c2}\right)\right) \end{aligned} \tag{4.5}$$

Basically what F2 measure does is to compute the overlapping region between the two classes (Fig. 4.8a). However, if we see Fig. 4.5a, it is obvious that even in low temperatures there are points of the opposite class near the boundary. And this is because of the periodic boundary conditions. In this manner, we don't expect to see something special by computing this measure. In Fig. 4.8b we can see F2 measure as a function of temperature. The behavior is trivial.



(a) (b)

Figure 4.8: (a) A scheme that help us to understand how we compute F2 measure, and (b) F2 measure as a function of temperature.

Maximum Individual Feature Efficiency (F3)

This measure estimates the individual efficiency of each feature in separating the classes and considers the maximum value found among the m features. The problem

can be considered simpler if there is at least one feature that shows low ambiguity between the classes, so F3 can be expressed as:

$$F3 = \min_{i=1}^{m} \frac{n_0(f_i)}{n} \tag{4.6}$$

where $n_0(f_i)$ gives the number of examples that are in the overlapping region for feature f_i,

$$n_0(f_i) = \sum_{j=1}^{n} I(x_{ji} > \max\min(f_i) \wedge x_{ji} < \min\max(f_i)) \tag{4.7}$$

This measure is very similar to F2, the only difference between them is that F2 compute areas, whereas F3 compute number of examples in overlapping region. Therefore, we expect the same trivial result that we had in F2 complexity measure (Fig. 4.9).

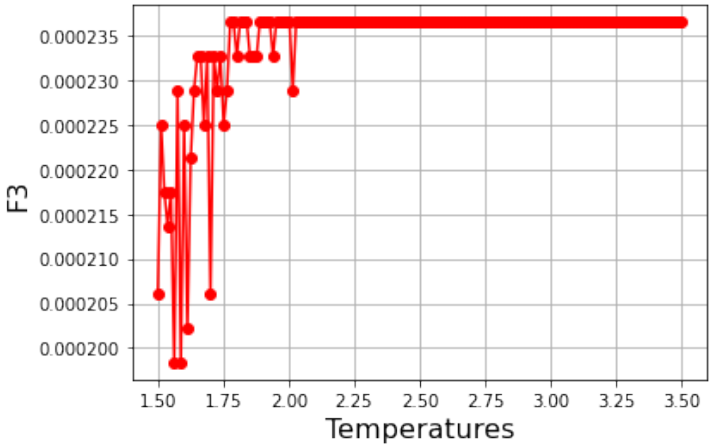


Figure 4.9: F3 measure for different temperatures (one configuration for each temperature). In this diagram we have 160 points on the interval $T \in [1.5, 3.5]$.

4.2.2 Measures of Linearity

To compute the next complexity measure we need to understand a bit of how Support Vector Machines (SVM) work. SVMs are linear classifiers that have as decision boundary an hyperplane that separates the examples between classes with a maximum margin and a minimum training errors. To find this hyperplane we must optimize the problem,

$$\text{Minimize}_{\boldsymbol{w},b,\epsilon} \frac{1}{2}||\boldsymbol{w}||^2 + C\left(\sum_{i=1}^{n} \epsilon_i\right) \tag{4.8}$$

$$\text{Subject to } \begin{cases} y_i(\boldsymbol{w} \cdot \boldsymbol{x_i} + b) \geq 1 - \epsilon_i \\ \epsilon_i \geq 0, i = 1, \ldots, n, \end{cases} \tag{4.9}$$

In this manner, in Fig. 4.10 we can see how an SVM works. We have these data in the boundary of each class, from which we want to maximize the margin, and sometimes we may have some miss-classified vectors. There are also kernel methods that allow us to have non-linear decision boundaries. However, here we don't need something like that.

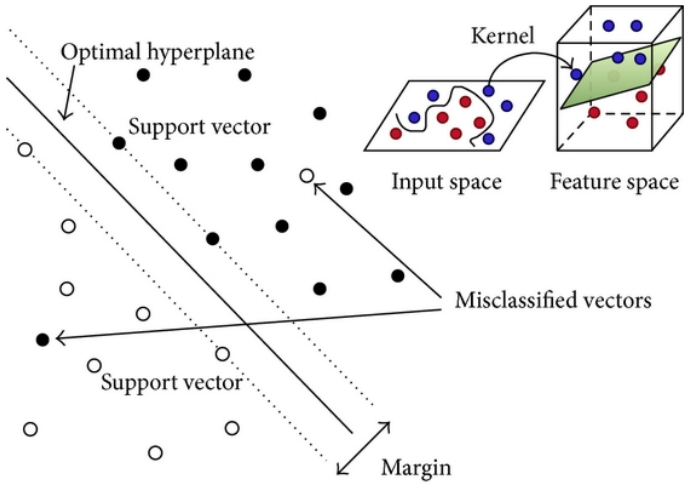


Figure 4.10: How SVM works.

Error Rate of Linear Classifier (L2)

The L2 measure computes the error rate of the linear SVM classifier. Let $h(x)$ denote the linear classifier obtained. L2 is then given by:

$$L2 = \frac{\sum_{i=1}^{n} I(h(\mathbf{x}_i) \neq y_i)}{n} \tag{4.10}$$

Higher L2 values denote more errors and therefore a greater complexity regarding the aspect that the data cannot be separated linearly. In Fig. 4.11 we see that this complexity measure give us another one trivial result[2]. Of course, it detects the critical point in some way, since we have a kind of sharp increase or discontinuity there, but it is still not what we want.

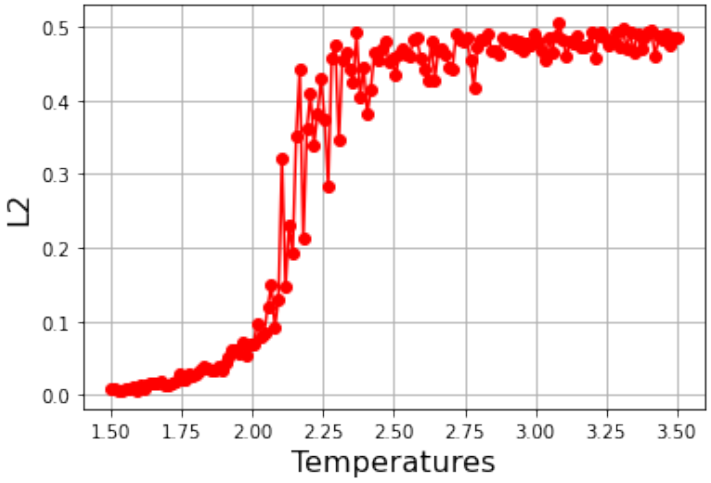


Figure 4.11: F3 measure for different temperatures (one configuration for each temperature). In this diagram we have 160 points on the interval $T \in [1.5, 3.5]$.

[2]We also computed the error with polynomial and rbf kernel, but we had the same trivial result.

4.2.3 Neighborhood Measures

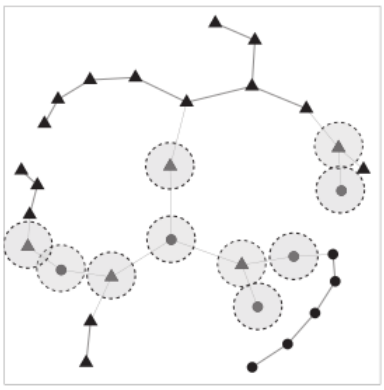
These measures try to capture the shape of the decision boundary and characterize the class overlap by analyzing local neighborhoods of the data points. All of them work over a distance matrix storing the distances between all pairs of points in the dataset.

Fraction of Borderline Points (N1)

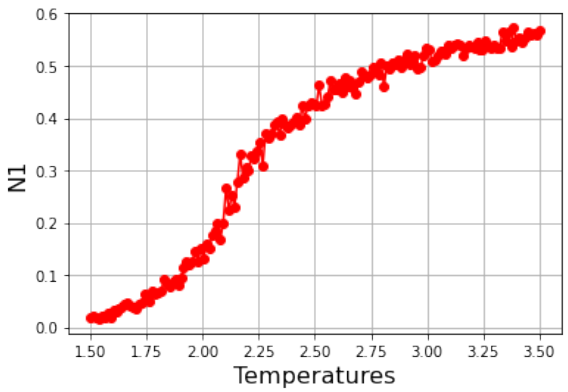
N1 is obtained by computing the percentage of vertices incident to edges connecting examples of opposite classes in the generated MST:

$$N1 = \frac{1}{n}\sum_{i=1}^{n} I((\mathbf{x}_i, \mathbf{x}_j) \in MST \wedge y_i \neq y_j) \tag{4.11}$$

To understand the way we compute MST, we have the Fig. 4.12a, where we see that this MST is the region where we have mixed classes in our classification problem. The behavior is still trivial: the decision boundary becomes bigger as the temperature increases. Moreover, we have not any kind of discontinuity in the critical point and we cannot say that it can detect the criticality in some way.



(a) MST



(b) N1 measure as a function of temperature.

Figure 4.12: (a) A scheme that help us to understand how we compute N1 measure, and (b) N1 measure as a function of temperature.

Error Rate of the Nearest Neighbor Classifier (N3)

The N3 measure refers to the error rate of a NN classifier that is estimated:

$$N3 = \frac{1}{n}\sum_{i=1}^{n} I(NN(\mathbf{x}_i) \neq y_i) \tag{4.12}$$

This is just another one measure that give us a similar result as the previous statistical measure. We computed the same measure for different number of nearest neighbors, and we had similar results.

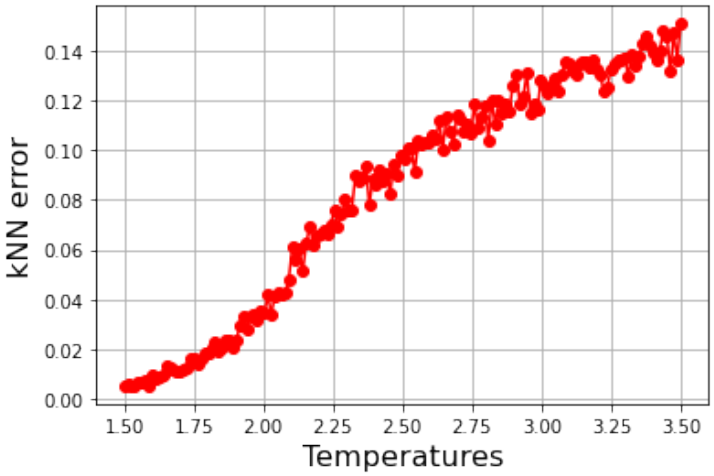


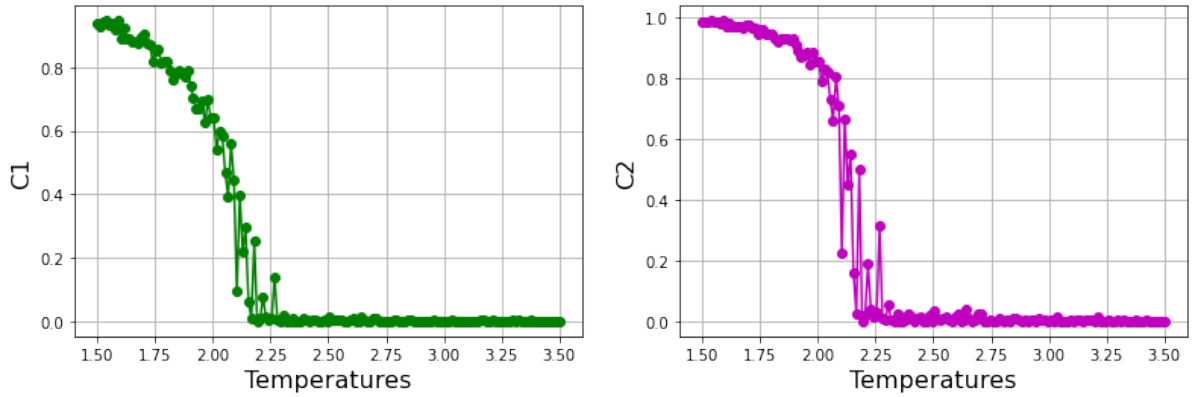
Figure 4.13: N3 measure as a function of temperature (for 3 nearest neighbors). In this diagram we have 160 points on the interval $T \in [1.5, 3.5]$.

4.2.4 Class imbalance measures

Here we compute two complexity measures:

- **Entropy of Class Proportions (C1):** $C1 = -\frac{1}{\log(n_{c_i})} \sum_{i=1}^{n_c} p_{c_i} \log(p_{c_i})$. This measure will achieve maximum value for balanced problems; that is, problems in which all proportions are equal. These can be considered simpler problems according to the class balance aspect.
- **Imbalance Ratio (C2):** $C2 = 1 - \frac{1}{IR}$, where

$$IR = \frac{n_c - 1}{n_c} \sum_{i=1}^{n} \frac{n_{c_i}}{n - n_{c_i}}.$$



(a) C1 Complexity Measure

(b) C2 Complexity Measure

Figure 4.14: C1 and C2 complexity measures as functions of temperature. In this diagram we have 160 points on the interval $T \in [1.5, 3.5]$.

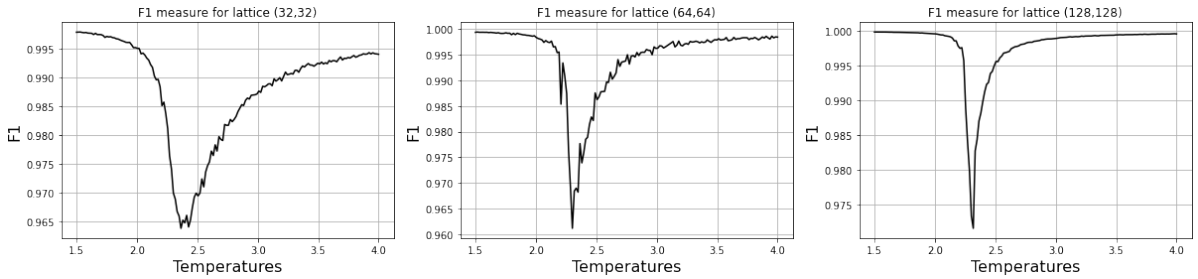
What this measure tell us is that in low temperatures our classes are imbalanced. This is because our lattice looks like 4.5a and we have much more spins from the one class than spins from the other one. This is another results that can detect the criticality, however it has not the behavior we want.

4.3 A Closer Look to F1 Complexity Measure

First of all, to reassure that F1 really is a complexity measure that can detect the criticality of Ising model. Therefore, there are two questions that we must answer: the first one is if this global minimum in criticality, is a kind of statistical fluctuation and the second one is about how global is this behavior - 'is this global minimum apparent for lower or higher lattice dimensions?'.

To answer the first question, we need to take the time average over the equilibrium states of Metropolis-Hastings algorithm. If we do that, we will have a general trent for the behavior of F1 measure for each temperature. To answer the second question, we need to run Metropolis-Hastings algorithm for different lattice sizes.

Consequently, we run Metropolis-Hastings algorithm for 10^8 steps, with sampling frequency 5×10^5. We compute F1 complexity measure for each sample in the equilibrium and we compute the mean value of F1 over time. We do that for the interval of temparatures $T \in [1.5, 4.0)$ (160 points in total). In Fig. 4.15 we see that the global minimum is always apparent. However, for small lattices the width of the dale is bigger than that in higher lattice dimensions. Additionally, we run the algorithm for $N = 256, 512$ and we saw that the results are still valid.



(a) $n = 32$ (b) $n = 64$ (c) $n = 128$

Figure 4.15: The average value of F1 over time for three different lattice sizes.

Apart from the temperature dependence of F1 measure, it is also interesting to have an idea about its fluctuations over the equilibrium of Metropolis-Hastings algorithm. In Fig. 4.16, we can see that these fluctuations are more intense in critical point than in low and high temperatures. This result remind us the behavior of magnetization (Fig. 4.4b). Indeed, if we see eq. 4.3 again, in the numerator there are means over the classes of positive and negative spins. These means are just sums over spins, and we can actually say that they are a kind of magnetization.

But the main question still exists: *why this global minimum in the critical point?* It is clear that these complexity measures give us a geometrical representation of criticality. Therefore, we should think geometrically. Let's see again the Fig. 4.5. We said that in critical point, our configuration becomes fractal as aggregates are shaped. These aggregates displace the center of mass (or the mean) of each class. Due to this displacement, the difference between the means of each class becomes maximum, and therefore F1 measure becomes minimum. In this regard, we can give *a very simple and clear interpretation* for the behavior of F1 measure.

This simplicity of F1 feature-based complexity measure, can be an inspiration to define some new complexity measures. In the next section we introduce some new

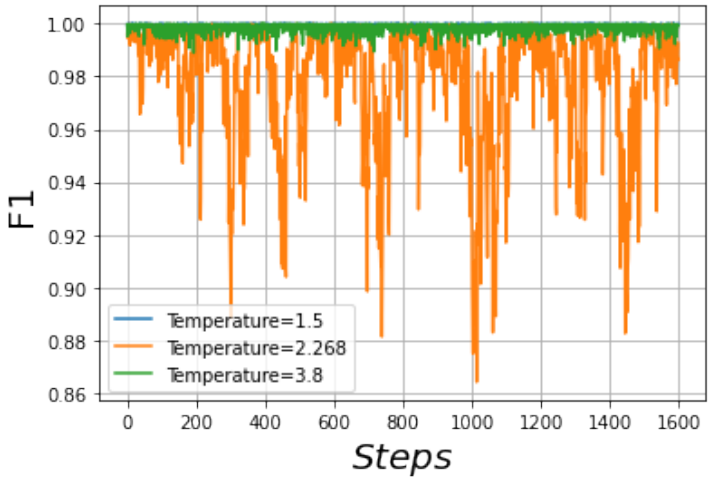


Figure 4.16: F1 measure fluctuates as the mean magnetization does.

complexity measures that look like F1, and all of them can detect the criticality of Ising model.

4.4 A Collection of New Complexity Measures

In this section we introduce some new complexity measures. To do that, we redefine F1 complexity measure to have an even simpler form in order to have a correspondence with the problem of Kullback-Leibler divergence minimization of Boltzmann Machines. Therefore, it is essential to do an introduction in Boltzmann Machines in order to have a clear picture about the connection between these measures and this Machine Learning algorithm.

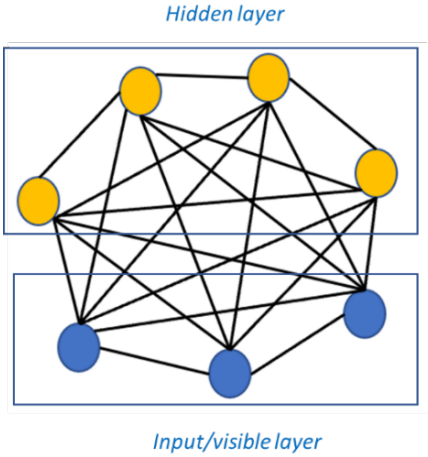


Figure 4.17: A Boltzmann Machine.

4.4.1 The inspiration: Boltzmann Machines

Boltzmann Machines (BM) are energy-based neural networks. The topology of these networks looks like a fully connected network with some visual and some hidden nodes (Fig. 4.17). We can see visual and hidden nodes as classes of our neural

network. This is a convenient topology in terms of complexity measures, because it remind is a bit this topology of Mean-Field approximation in Fig. 1.4, and thus we can think that there is an intuitive connection between these two representations, but in terms of Machine Learning, we usually choose to delete some of the connections between nodes and create the neural networks called Restricted Boltzmann Machines (RBM). In Fig. 4.18 we can see the topology of an RBM: we have two arrays of nodes (visual and hidden). There are no connections within the visual (or hidden) layer, but each visual node is connected with all the hidden nodes.

Energy Function

We said that BMs and RBMs are energy models, so which is the energy of these models? According to the source [31], the energy function of an RBM can be defined as,

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_i a_i u_i - \sum_j b_j h_j - \sum_{i,j} u_i h_j w_{ij} \tag{4.13}$$

where a_i and b_i are the biases which are related to visual and hidden layer respectively and w_{ij} are the weights that connect the two layers. Thereafter, we define the probability distribution of a certain state $(\boldsymbol{v}, \boldsymbol{h})$ as

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z_{\text{RBM}}} e^{-E(\boldsymbol{v},\boldsymbol{h})}, \qquad Z_{\text{RBM}} = \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \tag{4.14}$$

where Z_{RBM} is the partition function of RBM.

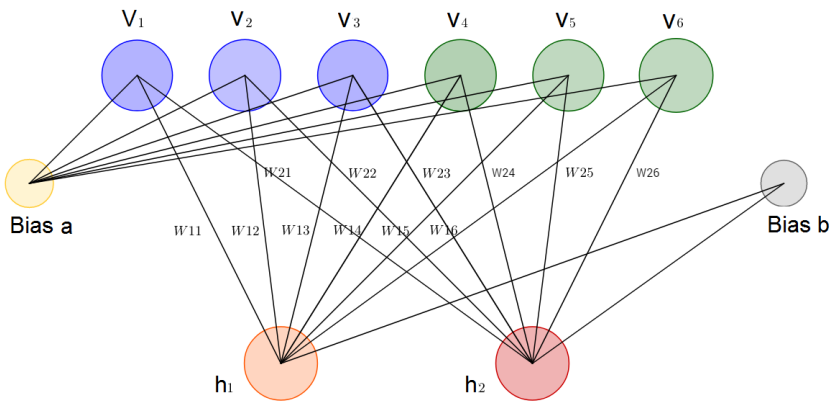


Figure 4.18: A Restricted Boltzmann Machine. ([31])

Sampling

And we can also define the conditional probability distributions,

$$p(\boldsymbol{h}|\boldsymbol{v}) = \frac{p(\boldsymbol{v},\boldsymbol{h})}{\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v})} \tag{4.15}$$

$$p(\boldsymbol{v}|\boldsymbol{h}) = \frac{p(\boldsymbol{v},\boldsymbol{h})}{\sum_{\boldsymbol{v}} p(\boldsymbol{h}|\boldsymbol{v})} \tag{4.16}$$

These probability distributions are useful for the training of RBMs. And from these two equations of conditional probabilities, we do a kind of Gibbs sampling to our network, viz instead of doing sampling from the distribution $p(\boldsymbol{v})$ in every iteration of our model, we compute, we correspond a visual layer $\boldsymbol{v}$ to a hidden one $\boldsymbol{h}$ via the conditional distribution $p(\boldsymbol{h}|\boldsymbol{v})$. On the other hand, the distribution $p(\boldsymbol{v}|\boldsymbol{h})$ does exactly the opposite procedure. Therefore, we can imagine a process that will correspond a visual sample, to the next most probable sample $\boldsymbol{v} \to \boldsymbol{v}'$.

Training

Also we can define the distribution $p(\boldsymbol{v})$ of the visual layer $\boldsymbol{v}$ as

$$p(\boldsymbol{v}) = \frac{1}{Z_{\text{RBM}}} \sum_{\boldsymbol{h}} e^{E(\boldsymbol{v},\boldsymbol{h})} \tag{4.17}$$

and due to this sum over the hidden states, it is equivalent to write

$$p(\boldsymbol{v}) \propto e^{H(\boldsymbol{v})}, \qquad H(\boldsymbol{v}) = -\sum_{i} a_i u_i - \sum_{i,j} u_i w_{ij} \tag{4.18}$$

In this manner to train a RBM model, we need an optimization function (like this mean square error function we saw in the polynomial regression example in the introduction of this thesis). Nevertheless, in Boltzmann Machines we don't do classification or regression, but we compare a theoretical probability distribution $q(\boldsymbol{v})$ with this distribution which is extracted from the algorithm $p(\boldsymbol{v})$. Kullback-Leibler (KL) divergence is a distance between two probability distributions,

$$KL(q||p) = \sum_{\boldsymbol{v}} q(\boldsymbol{v}) \log\left(\frac{q(\boldsymbol{v})}{p(\boldsymbol{v})}\right) \tag{4.19}$$

The main idea of this thesis is that KL divergence looks very similar to this KL divergence. The only difference is that for F1 the classes are the spins of our lattice, whereas for KL divergence the classes are a theoretical and a training example. This idea becomes clearer if we think that this KL divergence is proportional to a energy difference,

$$KL(q||p) = \sum_{\boldsymbol{v}} q(\boldsymbol{v}) \log\left(\frac{q(\boldsymbol{v})}{p(\boldsymbol{v})}\right) \propto \sum_{\boldsymbol{v}} q(\boldsymbol{v}) \left(H_{\text{model}}(\boldsymbol{v}) - H_{\text{data}}(\boldsymbol{v})\right) \tag{4.20}$$

This is a very essential idea to define our new complexity measures. Note also that all the nodes in BMs can take only two values 0 and 1. This is another very essential part, because it allow us to think Boltzmann Machines as lattices. Perhaps this idea is

just intuition, but it can help us to define some new complexity measures. However, we will come back to it later.

To train a BM, we need to make our network to follow the direction where our optimization function becomes minimum. And the way we do that is by computing the derivatives of KL divergence,

$$w_{ij} \leftarrow w_{ij} - \epsilon \frac{\partial KL}{\partial w_{ij}} \tag{4.21}$$

$$a_i \leftarrow a_i - \epsilon \frac{\partial KL}{\partial a_i} \tag{4.22}$$

$$b_i \leftarrow b_i - \epsilon \frac{\partial KL}{\partial b_i} \tag{4.23}$$

where ϵ is the learning rate of BM.

Contrastive Divergence

The problem with BMs is that this optimization problem is unprofitable in terms of computational time due to the high dimensionality of the state space $(\boldsymbol{v}, \boldsymbol{h})$. Therefore, there is an alternative approach that help us to save computational time by using the idea of Expectation Maximization (EM). So we said that we do sampling from these conditional probability distributions,

$$p(\boldsymbol{h}|\boldsymbol{v}) = \prod_i p(h_i|\boldsymbol{v}) = \prod_i \frac{e^{h_i(b_i + \sum_j u_j w_{ji})}}{2\cosh\left(b_i + \sum_j u_j w_{ji}\right)} \tag{4.24}$$

$$p(\boldsymbol{v}|\boldsymbol{h}) = \prod_i p(v_i|\boldsymbol{h}) = \prod_i \frac{e^{u_i(a_i + \sum_j h_j w_{ji})}}{2\cosh\left(a_i + \sum_j h_j w_{ji}\right)} \tag{4.25}$$

Thereby, from $p(h_i|\boldsymbol{v})$ we estimate hidden layer nodes $\hat{h}_i$, and then we can estimate nodes $\tilde{u}_i$ from $p(u_i|\boldsymbol{h})$ given the sample $\hat{\boldsymbol{h}}$. And thus we can estimate the mean values,

$$\langle u_i h_j \rangle_{\text{data}} = \frac{1}{N} \sum_{i,j} \hat{u}_i \hat{h}_j, \qquad \langle u_i h_j \rangle_{\text{model}} = \frac{1}{N} \sum_{i,j} \tilde{u}_i \tilde{h}_j \tag{4.26}$$

$$\langle u_i \rangle_{\text{data}} = \frac{1}{N} \sum_{i} \hat{u}_i, \qquad \langle u_i \rangle_{\text{model}} = \frac{1}{N} \sum_{i} \tilde{u}_i \tag{4.27}$$

$$\langle h_i \rangle_{\text{data}} = \frac{1}{N} \sum_{i} \hat{h}_i, \qquad \langle h_i \rangle_{\text{model}} = \frac{1}{N} \sum_{i} \tilde{h}_i \tag{4.28}$$

where N is the number of all the possible samples. In these terms, we update the weights using the following rule,

$$w_{ij} \leftarrow w_{ij} + \epsilon(\langle u_i h_j \rangle_{\text{data}} - \langle u_i h_j \rangle_{\text{model}}) \tag{4.29}$$

$$a_i \leftarrow a_i + \epsilon(\langle u_i \rangle_{\text{data}} - \langle u_i \rangle_{model}) \tag{4.30}$$

$$b_i \leftarrow b_i + \epsilon(\langle h_i \rangle_{\text{data}} - \langle h_i \rangle_{model}) \tag{4.31}$$

this rule is what is called *'contrastive divergence'*.

RBM flow

Having trained our network, we end up with some estimations for weights w_{ij} and biases a_i and b_i, which are constants. So having a specific configuration in our input nodes $\boldsymbol{v}_1$ we can do Gibbs sampling and produce a new visual configuration $\boldsymbol{v}_2$. If we repeat the process for k steps, we would end up in a configuration $\boldsymbol{v}_k$, as Fig. 4.19 shows. The process of doing these k reconstruction steps is called RBM flow.

If we feed a Boltzmann Machine with configurations of Ising model in different temperatures, that include critical temperature, and train our machine, then we have a very paradoxical behavior. Because having trained the machine, if we do a big enough amount of flow steps $k \to \infty$, then we will always end up to configurations that remind us Ising configurations in critical region. According to [32], [33], the probability distribution of configurations that we end up to after the flow, looks pretty similar to this of Metropolis-Hasting's algorithm near the critical region.

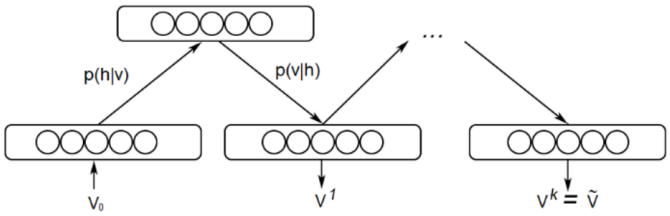


Figure 4.19: RBM flow

4.4.2 Redefining F1 complexity measure

After this long introduction to BMs, we can think how to redefine F1 complexity measure, so as to have a relation with this algorithm. We see that in our algorithm as it is described above, we compute some differences between some *'data'* or *'model'* samples, and these are basically two classes of nodes. Also we observe that our nodes take binary values and actually the energy of BMs looks pretty similar to the energy of the Ising model. The main difference between the two energies is that in BMs, we have an extra complexity which is related to the weights and biases of our machine.

First of all, in this section we work only with Ising model, which means that we don't run any Machine Learning algorithm. Therefore, we can think in two ways: we can split one Ising lattice in two parts or we can compare two lattices. We name the one part/lattice *'input layer'*, and the other one *'hidden layer'*. It doesn't matter the name, we could call it *'data'* and *'model'* if we liked to, but we preferred to call it in this way.

Let's denote input layer spins as $\boldsymbol{v}$ and hidden spins as $\boldsymbol{h}$. In this manner, we redefine F1 measure as Seb Index (SI),

$$r = \frac{(|\langle \boldsymbol{v} \rangle| - |\langle \boldsymbol{h} \rangle|)^2}{2(Var(\boldsymbol{v}) + Var(\boldsymbol{h}))}, \quad SI = \frac{1}{1+r} \tag{4.32}$$

And the global minimum still exists, if we assume the input and hidden layer are in the same cropped image. So we can run an Ising model for 160 different temperatures in the interval $T \in [1.5, 3.5]$. For each temperature we take the equilibrium configurations, split them in two equal parts (left and right) and compute this SI complexity

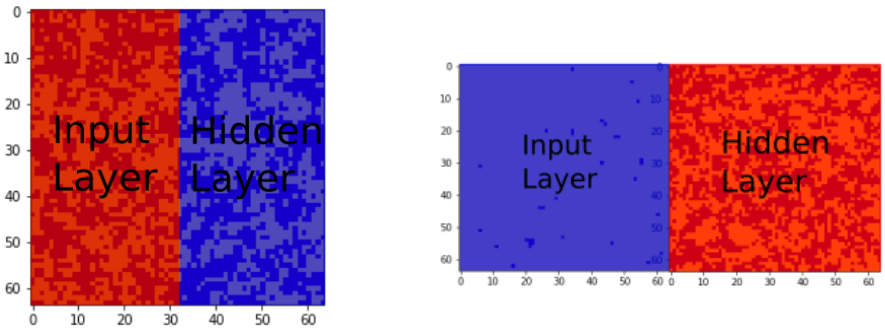


Figure 4.20: We can think in two ways: to split one lattice in two parts (image on the left), or to compare two different lattices (image on the right).

measure for each equilibrium configuration of Metropolis-Hastings simulation. We do 10^8 steps with sampling frequency 5×10^5. In Fig. 4.21 we see the time average of this SI complexity measure, and it is clear that we succeeded to create a measure that detects the criticality. The reason why we still have this global minimum, is because of the fluctuations of magnetization and the break of symmetry (viz the aggregates that are shaped).

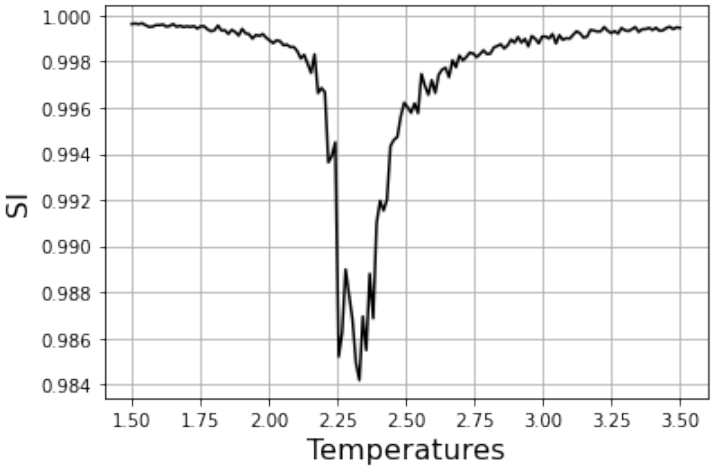
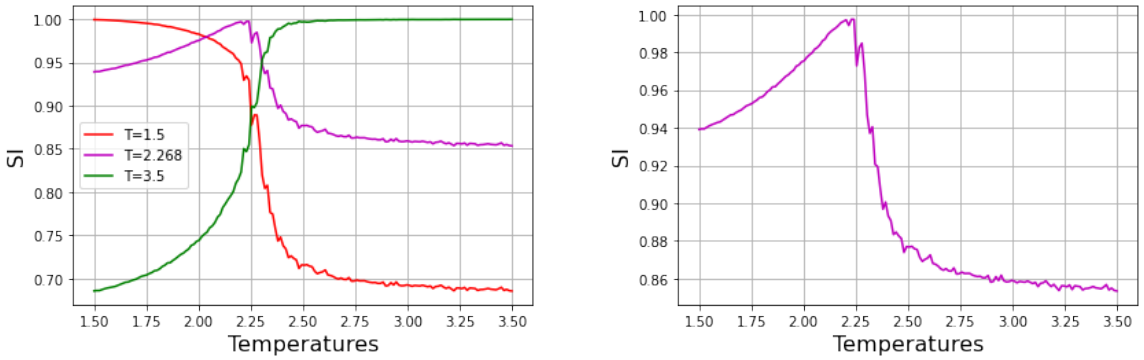


Figure 4.21: SI as a function of temperature of Ising model, for the same input and hidden layer configuration.

Now let's try to do something else: instead of computing SI for the same lattice, we can take three fixed equilibrium configurations in hidden layer and change the temperature of input layer. In input layer, we take the time averages of each equilibrium configuration. In Fig. 4.22 we can see that the shape of isothermic curves changes in respect to the hidden layer temperature. And the most interesting part of this kind of phase transition is what happens in the critical point, where the isothermic curve of SI complexity measure has a clear global maximum. This SI measure is inversely proportional to the difference of means of the two probability distributions, which means that the isothermic curve of the difference of the first moments of these two probability distributions has a global minimum in the critical temperature.



(a) Varying temperatures in hidden layer.

(b) Hidden layer in critical temperature.

Figure 4.22: SI Complexity Measure for different input and hidden layer configurations.

4.4.3 Difference of Means (DoM)

We define the difference of means (DoM) as,

$$DoM = (|\langle \boldsymbol{v} \rangle| - |\langle \boldsymbol{h} \rangle|)^2 \tag{4.33}$$

This is perhaps the simpler complexity measure between input and hidden layer we can define. However, it is very essential for our research because it has the first order information of the difference of probability distribution between our classes.

From Fig. 4.23 it clear that if we plot DoM as a function of temperature for the same lattice, it has a global maximum at the critical point.

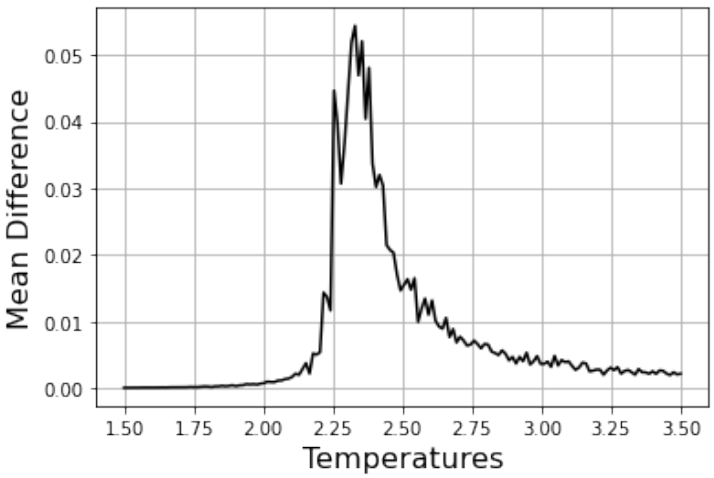
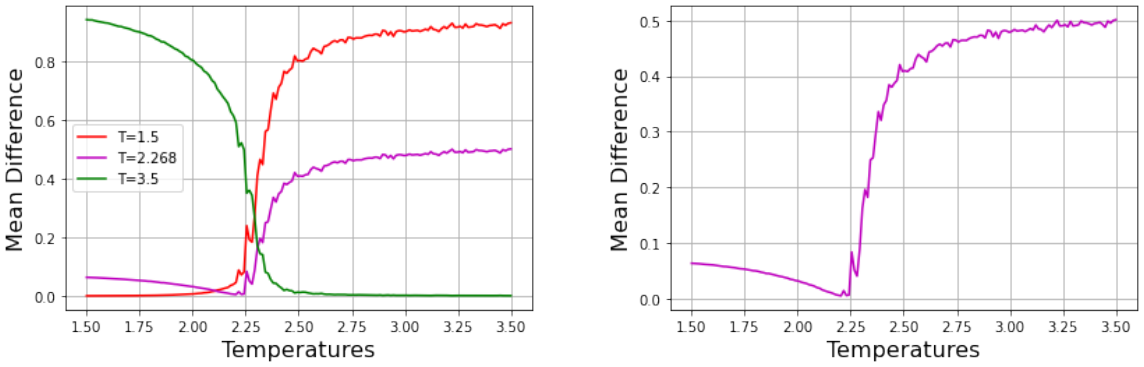


Figure 4.23: DoM as a function of temperature of Ising model, for the same input and hidden layer configuration.

On the other hand, if we plot DoM as a function of temperature for hidden layer lattice configurations fixed in three different temperatures, we see that the isothermic curves have a global minimum when our hidden layer is in the critical point (Fig. 4.24). If the hidden layer is not in the critical point, there is not a global minimum at the critical point, but the critical region is still accessible. When we say that it is *accessible*, we mean that there is always a way to change the hidden layer so as to approach the critical region and keep DoM as small as possible.



(a) Varying temperatures in hidden layer.

(b) Hidden layer in critical temperature.

Figure 4.24: DoM complexity measure for different input and hidden layer configurations.

For example, if we assume that our hidden layer is in low temperatures, and we have an optimization problem that tends to minimize DoM, then our hidden layer will change so as to have less or equal DoM with the previous one. Therefore, assuming that all the temperatures which are $T < T_c$ are accessible, we can always reach critical temperature from the left. And if we reach the critical region, the curve closes and the system cannot escape from this region. Therefore, this example offer us a new idea about Boltzmann Machines: to use DoM as optimization function instead of KL divergence. Note that in Boltzmann Machines we cannot assume that in hidden layer there are always Ising lattices, and this means that the previous thoughts need more research.

4.4.4 Difference of Variances (DoV)

Similarly, we define the difference of variances as,

$$DoV = (Var(\boldsymbol{v}) - Var(\boldsymbol{h}))^2 \tag{4.34}$$

and we can see that this complexity measure can also detect the critical temperature of Ising model (Fig. 4.25).

Additionally, we can plot the isothermic curves of DoV for three different fixed hidden layer configurations, and we see that the behavior looks pretty similar to the behavior of DoM complexity measure (Fig. 4.26). The global minimum for the critical hidden configuration is very clear here.

4.4.5 Difference of Skewnesses and Kurtoses (DoS, DoK)

We can also check what happens with third and fourth order statistics and define,

$$DoS = (Sk(\boldsymbol{v}) - Sk(\boldsymbol{h}))^2 \tag{4.35}$$

$$DoK = (Ku(\boldsymbol{v}) - Ku(\boldsymbol{h}))^2 \tag{4.36}$$

where Sk and Ku are the third and fourth order moments[3], which are called skewness and kurtosis respectively. However, if we try to create similar plots (Fig. 4.27) we see

[3]See section 3.2 for a more detailed definition and the explanation of their properties.

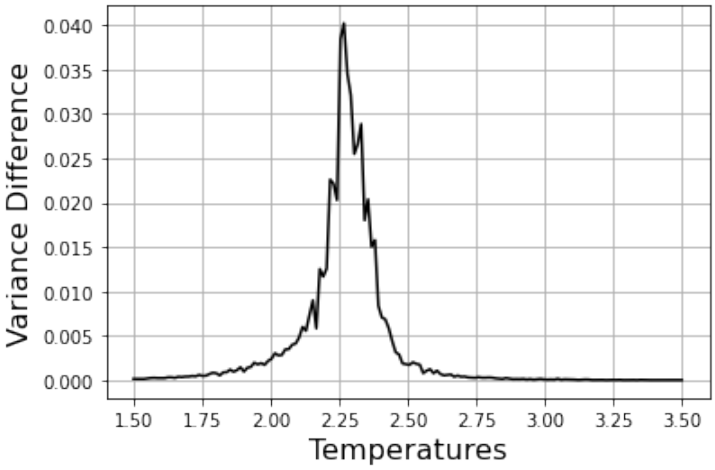
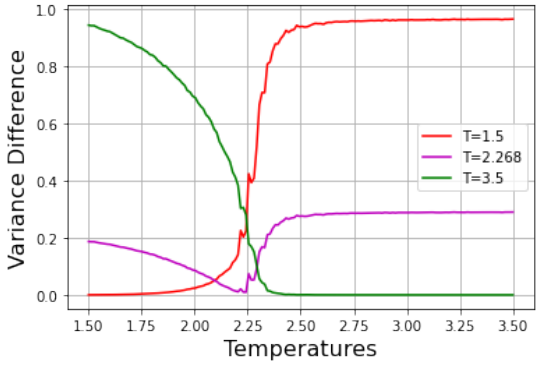
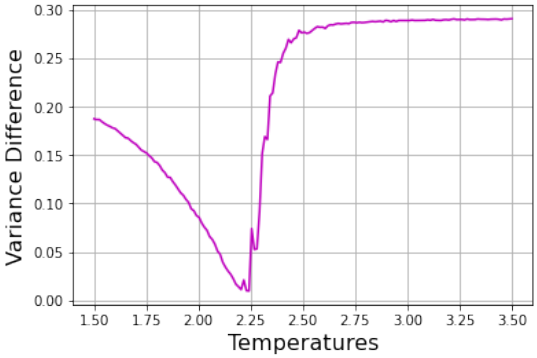


Figure 4.25: DoV as a function of temperature of Ising model, for the same input and hidden layer configuration.



(a) Varying temperatures in hidden layer.



(b) Hidden layer in critical temperature.

Figure 4.26: DoV complexity measure for different input and hidden layer configurations.

that the interesting behavior disappears. These statistical measures are unable to detect the criticality, and there is not any global minimum when the hidden layer is in critical temperature.

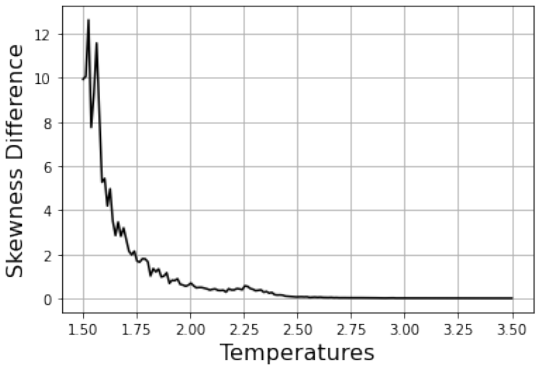
4.4.6 KL Divergence

We can also define a KL Divergence Complexity Measure. However, this is a different complexity measure from the KL Divergence in BMs. In BMs we have some probability distributions that depend on the weights and biases of the machine. Here we want to avoid the computation of biases and weights, provided that our main purpose is to create complexity measures that can detect the criticality. The way we define a probability distribution for a lattice S is

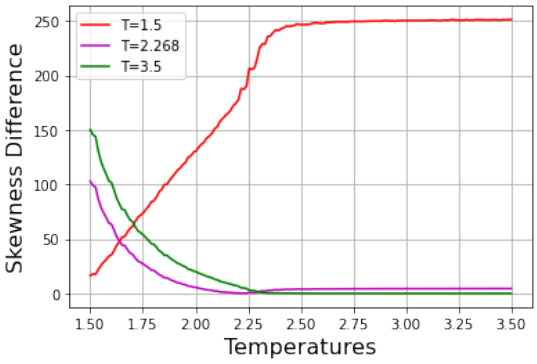
$$P(s_i) = [p(s_i = +1), p(s_i = -1)] \tag{4.37}$$

and then we compute

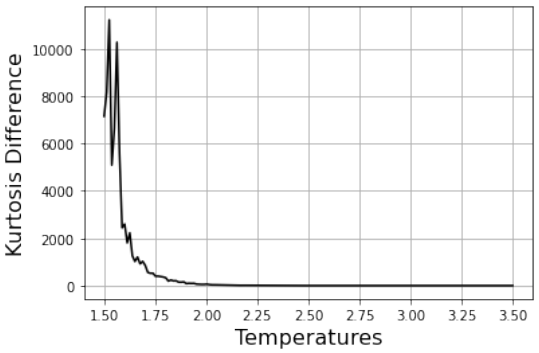
$$KL(q||p) = \sum_k P_{\text{input}}(k) \log \left(\frac{P_{\text{input}}(k)}{P_{\text{hidden}}(k)} \right) \tag{4.38}$$



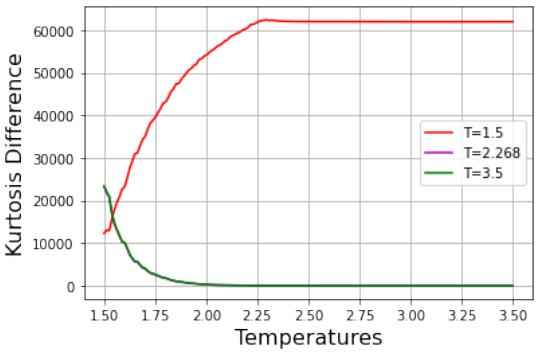
(a) Same hidden and input configuration.



(b) Hidden layer in three temperatures.



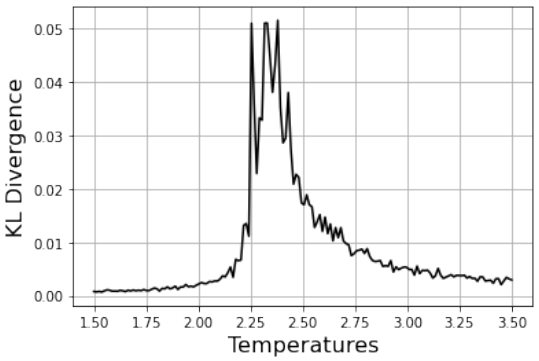
(c) Same hidden and input configuration.



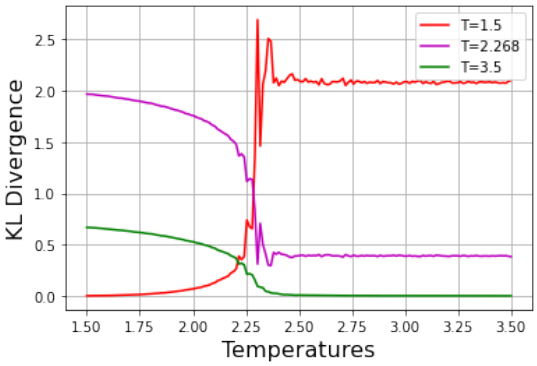
(d) Hidden layer in three temperatures.

Figure 4.27: DoS, DoK complexity measure.

where these probability distributions $P(\cdot)$ can be computed for both input or hidden layer nodes. In Fig. 4.28 we see that this complexity measure can also detect the critical temperature, if we split each lattice in two parts. If we compare two different lattices, there is still a kind of phase transition in the shape of isothermic curves, but it is not so clear if there is any global minimum when the hidden layer is in critical point.



(a) Same hidden and input configuration.



(b) Hidden layer in three temperatures.

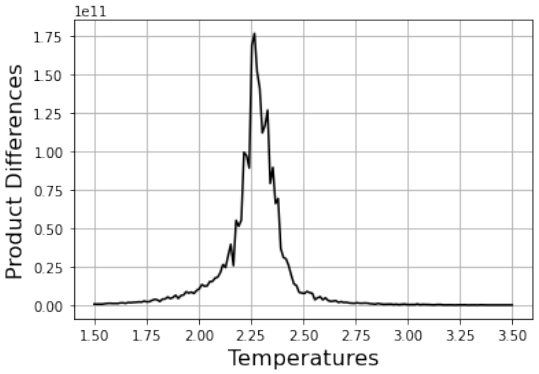
Figure 4.28: KL divergence complexity measure.

4.4.7 Difference of Pair Products (DoP)

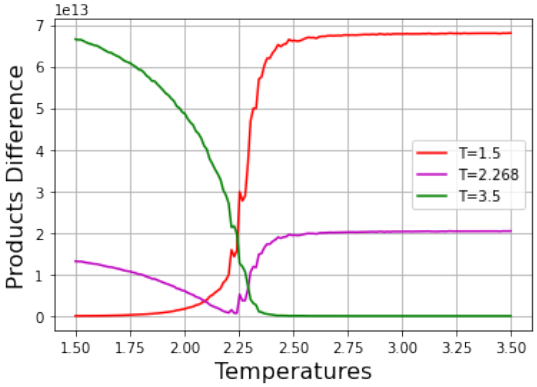
When we write the Hamiltonian of Ising model (or the energy function of BMs) we have two main term: one term which is related to an external magnetic field B and basically is a sum over spins $\sum_i s_i$ and another one which is related to interactions of spins $\sum_{<ij>} s_i s_j$ (or in case of BMs $\sum_{i \neq j} s_i s_j$). We have analyzed the first term, but we haven't say anything about the second one yet. Therefore, we define a new complexity measure, which we call *difference of (pair) products (or just DoP)*,

$$DoP = \left(\left| \sum_{i \neq j} u_i u_j \right| - \left| \sum_{i \neq j} h_i h_j \right| \right)^2 \tag{4.39}$$

This complexity measure can detect the criticality (4.29). Another advantage of this measure is that has a global minimum when the hidden layer is in the critical temperature. Note that this behavior is apparent in both DoM and DoP, which are the two main factors of our energy function.



(a) Same hidden and input configuration.



(b) Hidden layer in three temperatures.

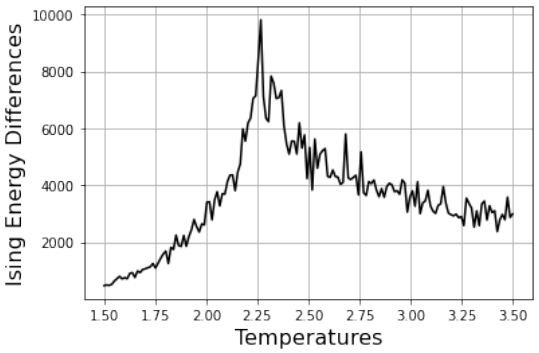
Figure 4.29: DoP complexity measure.

4.4.8 Difference of Ising Energies (DoE)

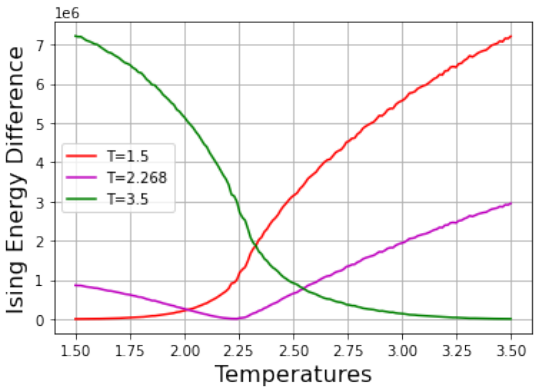
Moreover, we can compute something like the difference of energies, if we assume the Ising energies of each lattice. The way we compute these energies is by using the equation eq. 1.1 and summing all the energy differences for each spin to flip.

$$\begin{aligned} DoE &= (E_{\text{input}} - E_{\text{hidden}})^2 \\ &= \left(\left| -J \sum_{<ij>} u_i u_j - B \sum_i u_i \right| - \left| -J \sum_{<ij>} h_i h_j - B \sum_i h_i \right| \right)^2 \end{aligned} \tag{4.40}$$

In Fig. 4.30 we can see that the behavior is pretty similar to DoP's. And basically, this is very reasonable, since we have assumed that $B = 0$. The only reason why we see a bit different numbers is because in DoP the sum runs in all spins for which $i \neq j$, whereas in Ising model we sum only the nearest neighbor spins $< ij >$. However, this complexity measure still detects the critical point, since we have a maximum for this temperature, and also the isothermic curves have the same behavior as DoP's.



(a) Same hidden and input configuration.



(b) Hidden layer in three temperatures.

Figure 4.30: DoE complexity measure.

4.5 Conclusions

In this Chapter we defined some classification complexity measures, and we saw that one of them (F1) can detect the criticality of Ising model. Additionally, we did a parallelism with Boltzmann Machines to define some new complexity measures. Specifically, we saw that it is very easy to construct complexity measures that can detect the critical point, if we split an Ising lattice configuration into two parts (left and right). And this happens because the symmetry of our system breaks in critical region and we have very intense fluctuations in magnetization. Now the main question is *'why we need these complexity measures?'* and *'what we will do with them in the next chapters?'*.

This research of complexity measures is essential because,

1. We can use them as diagnostics to know if a Metropolis-Hastings algorithm[4] run correctly.

2. They give us a better understanding about the Ising model criticality. For instance, when we defined F1 complexity measure we saw that it detects the criticality because aggregates are shaped and displace the centers of mass of the two classes.

3. Perhaps they can be used to understand the behavior of BMs. The fact that we see these global minima in some complexity measures, may make us suspicious. Somebody can assume that KL Divergence of BMs is just another complexity measure, whose isothermic curves are subject to phase transitions in order to have a global minimum when the hidden layer distribution is in the critical point, and maybe this is the reason why when we feed them with Ising lattices and do the flow, it always falls at a point in critical region. However, this thought needs more investigation, because we should not forget the weights and biases of BMs.

[4] In the next chapter, we create an alternative algorithm to Metropolis-Hastings with Machine Learning techniques. Therefore, in this case it is essential to have some diagnostics to see if it runs correctly.

To get back up to the shining world from there
My guide and I went into that hidden tunnel,

And Following its path, we took no care
To rest, but climbed: he first, then I-so far,
through a round aperture I saw appear

Some of the beautiful things that Heaven bears,
Where we came forth, and once more saw the stars.

- Dante Alighieri, Inferno

5 Searching For The Mapping

In this section we investigate the connection between Gaussian surfaces and Ising model. Our main aim is to find a systematic way to reconstruct the patterns of Ising model. In this manner, it is important not only to produce lattices with similar geometrical structure, but also we should reconstruct the fluctuations of magnetization. To do that we would use Machine Learning techniques to do a kind of interpolation so that to predict the parameters of Gaussian surfaces given a specific temperature. Consequently, we need to find the parametric path we should follow to reconstruct the patterns of Ising model.

5.1 Machine Learning Algorithms

If we really want to use Machine Learning to reach our goal, we should do a small introduction to have a basic understanding about the algorithms that we will see. In this section we mainly work with regression (not classification), and thus the known algorithms of linear and polynomial regression are also useful. However, it makes sense to check the power of other algorithms like Support Vector Machines, Decision Trees, Nearest Neighbors, Random Forests or Multi-layer Perceptron.

5.1.1 Linear Regression

Let's assume that we have some data like these in Fig. 5.1. It is clear that there is a linear dependence between the features on y_i and x_i axis. The model of linear regression is the basis of the science of Statistics, and thereby the basis of Data Science and Machine Learning. Five assumptions must be satisfied to do linear regression [59],

1. **Weak exogeneity:** which means that that the variables x_i are treated as fixed values and error-free.

2. **Linearity:** which means that we can write the response variable y as a linear combination of the parameters of our model (the coefficients and the intercept of the line) and the predictor variables x_i.

3. **Constant variance:** This means that the variance of errors does not depend on the variables x_i. In this manner, the total sum of errors ϵ (viz the distances of points from the line) must be zero.

4. **Independence of errors:** which means that errors ϵ must be uncorrelated.

5. **Lack of perfect multicollinearity:** which means that the features/predictors x_i must not be linearly dependent. Or alternatively, the matrix of X must have full column rank p.

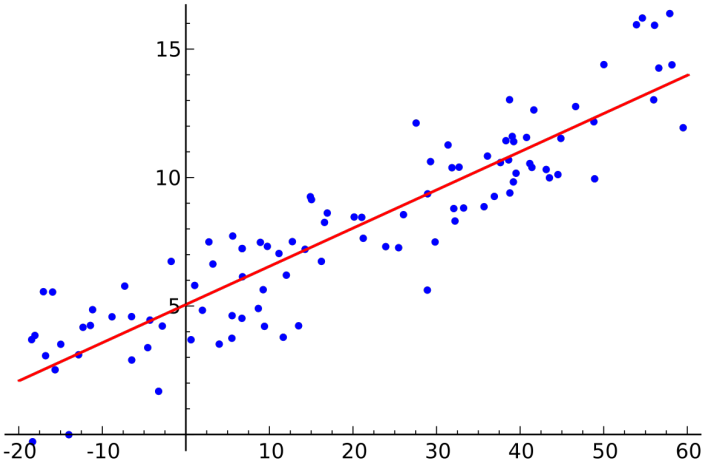


Figure 5.1: An example of linear regression.

In this manner, we can write the linear regression model as,

$$\boldsymbol{y} = X\boldsymbol{w} + \boldsymbol{\epsilon} \tag{5.1}$$

where $\boldsymbol{x}$ are the coefficients and $\boldsymbol{\epsilon}$ are the errors or residuals of linear regression. The loss function of the model is the sum of squared errors,

$$\begin{aligned} L(\boldsymbol{w}) &= \sum_{i=1}^{N} \left(y_i - \boldsymbol{w}^T \boldsymbol{x_i}\right)^2 = \sum_{i=1}^{N} (\epsilon_i)^2 \\ &= \begin{pmatrix} \epsilon_1 & \epsilon_2 & \dots & \epsilon_N \end{pmatrix} \begin{pmatrix} \epsilon_1 \\ \epsilon_2 \\ \vdots \\ \epsilon_N \end{pmatrix} = \boldsymbol{\epsilon}^T \boldsymbol{\epsilon} \end{aligned} \tag{5.2}$$

and to find the coefficients of linear regression we need to minimize the loss function $L(\boldsymbol{w})$ in respect to $\boldsymbol{w}$. If we do that, we can estimate the coefficients as,

$$\boldsymbol{w} = (X^T X)^{-1} X^T \boldsymbol{y} \tag{5.3}$$

5.1.2 Polynomial Regression

Polynomial regression is just a more generalized form of linear regression [21]. Assuming that we can illustrate $\boldsymbol{x}$ to an $\boldsymbol{\phi(x)}$ so that,

$$\boldsymbol{x} \rightarrow \boldsymbol{\phi(x)} = \begin{pmatrix} \phi_1(\boldsymbol{x}) \\ \phi_2(\boldsymbol{x}) \\ \vdots \\ \phi_M(\boldsymbol{x}) \end{pmatrix}, \tag{5.4}$$

so that the inner product,

$$\langle \boldsymbol{w}, \boldsymbol{\phi(x)} \rangle = w_0 + w_1 x + \cdots + w_k(x)^K. \tag{5.5}$$

In this case, we define our loss function similarly,

$$L(\boldsymbol{w}) = \sum_{i=1}^{N} (y^i - \boldsymbol{w^T x^i})^2 = \sum_{i=1}^{N} (\epsilon^i)^2 \tag{5.6}$$

and we can write our regression problem for more dimensions in $\boldsymbol{x}$ in matrix form as,

$$\begin{pmatrix} y^1 \\ y^2 \\ \vdots \\ y^N \end{pmatrix} = \begin{pmatrix} x_1^1 & \dots & x_D^1 \\ x_1^2 & \dots & x_D^2 \\ \vdots & \ddots & \vdots \\ x_1^N & \dots & x_D^N \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_D \end{pmatrix} + \begin{pmatrix} \epsilon^1 \\ \epsilon^2 \\ \vdots \\ \epsilon^N \end{pmatrix}. \tag{5.7}$$

And to find the coefficients of the generalized regression model,

$$\boldsymbol{y} = \boldsymbol{\Phi w} + \boldsymbol{\epsilon} \tag{5.8}$$

$$L(\boldsymbol{w}) = \boldsymbol{\epsilon^T \epsilon} \tag{5.9}$$

we just need to solve a similar equation as linear regression's one,

$$\boldsymbol{w}^* = \left(\boldsymbol{\Phi^T \Phi}\right)^{-1} \boldsymbol{\Phi^T y}. \tag{5.10}$$

5.1.3 Support Vector Machine Regression

According to [52], assuming that we have training data $\{(x_1, y_1), \dots, (x_l, y_l)\} \in \mathcal{X} \times \mathbb{R}$, we are searching for a linear function $f(x)$ that has at most ϵ deviation from the actually obtained targets y_i for all the training data and at the same time to be as flat as possible. Therefore, we start from linear functions of the form,

$$f(x) = \langle w, x \rangle + b \text{ with } w \in \mathcal{X},\ b \in \mathbb{R} \tag{5.11}$$

and we need to satisfy the constraints,

$$\begin{aligned} &\text{minimize} && \frac{1}{2}||w||^2 \\ &\text{subject to} && \begin{cases} y_i - \langle w, x_i \rangle - b \leq \epsilon \\ \langle w, x_i \rangle + b - y_i \leq \epsilon \end{cases} \end{aligned} \tag{5.12}$$

In this manner, our decision boundary depends on the choice of ϵ and if we want to illustrate it, it looks like this in Fig. 5.2. However, apart from linear SVMs there are

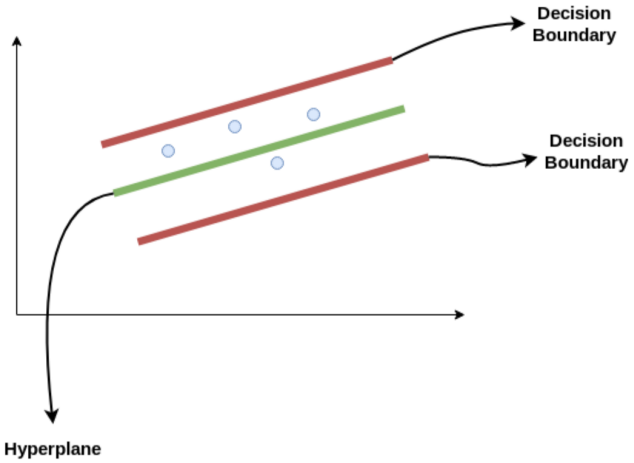


Figure 5.2: Support Vector Machine Regression.

also non-linear SVMs, and the only thing we must do to produce them is a mapping $\Phi : \mathcal{X} \to \mathcal{F}$, into some feature space $\mathcal{F}$. In SVM models we usually do the transformation via the kernel $k(x, x') = \langle \Phi(x), \Phi(x') \rangle$ and we speak about *'kernel methods'*. The main reason we do this, is because it saves computational time to work with inner products.

5.1.4 k-Nearest Neighbors Regression

This is perhaps the simpler regression algorithm. It is a nonlinear regressor and there is not any formula in closed form for it. What it does is to predict average label of k closest neighbors [53]. Assuming that $X^{(i)}(x)$ is the i-th nearest neighbor of a

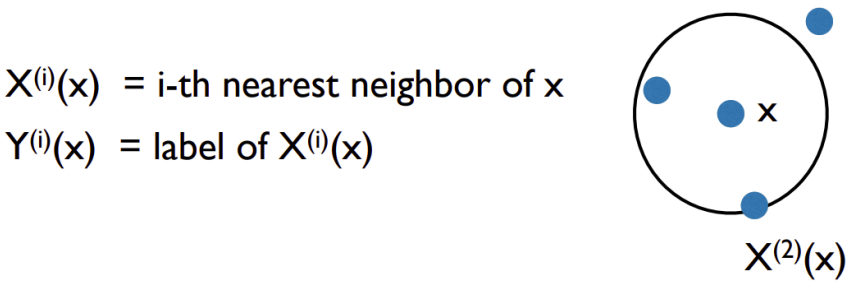


Figure 5.3: k-Nearest Neighbors Regression Regression. ([53])

given x and $Y^{(i)}(x)$ is the label of $X^{(i)}(x)$ (as it shown in Fig. 5.3), then we can write,

$$y_i = f(x_i) + \text{noise} \tag{5.13}$$

where the function $f(x_i)$ is unknown. And the way we estimate the values of the function $f(\cdot)$ for each x_i is by the average,

$$\hat{f}_k(x) = \frac{1}{k} \sum_{i=1}^{k} Y^{(i)}(x). \tag{5.14}$$

5.1.5 Decision Tree Regression

Decision Trees are also non linear regressors which don't have a formula in closed form. These kind of algorithms have their own terminology and thereby it is important to know it [55],

1. **Root Node:** It represents entire population or sample and this further gets divided into two or more homogeneous sets.
2. **Splitting:** It is a process of dividing a node into two or more sub-nodes.
3. **Decision Node:** When a sub-node splits into further sub-nodes, then it is called decision node.
4. **Leaf/Terminal Node:** Nodes do not split is called Leaf or Terminal node.
5. **Pruning:** When we remove sub-nodes of a decision node, this process is called pruning. You can say opposite process of splitting.
6. **Branch / Sub-Tree:** A sub section of entire tree is called branch or sub-tree.
7. **Parent and Child Node:** A node, which is divided into sub-nodes is called parent node of sub-nodes whereas sub-nodes are the child of parent node.

In Fig. 5.4 we can see a schematic representation of a decision tree and all the related terminology. The way a decision tree works is by asking a series of questions to

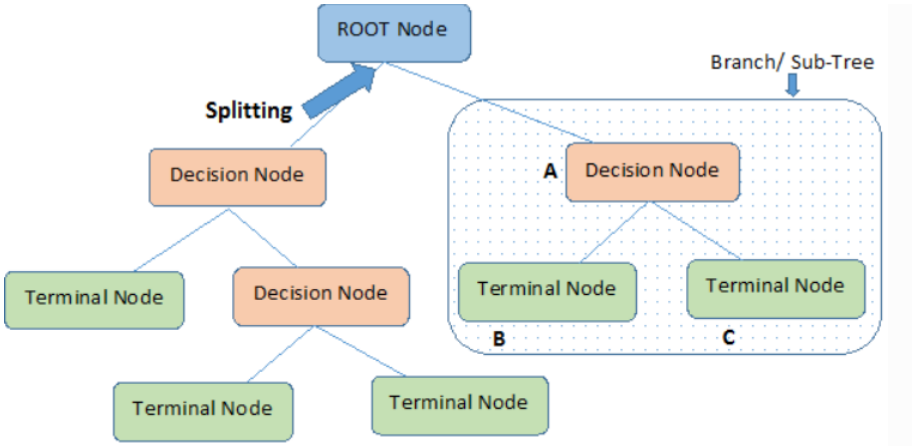


Figure 5.4: A decision tree. ([55])

the data. For each question there are only binary answers. Our tree keeps asking questions, until it will get confident enough to make a single prediction. In this manner, if we assume that we have two features x_1 and x_2, we start from a root node and we start asking questions. The answers separate nodes in branches. The decision tree learns these questions and the thresholds of the questions.

5.1.6 Random Forest Regression

This is another decision-tree-like algorithm. The philosophy here is that instead of training one tree and do predictions with that, we train a whole forest of decision trees and then we keep the average prediction (Fig. 5.6). It is one of the most popular non linear regressors due to its power and simplicity. Therefore, if we want to summarize its function in two steps, it is [58],

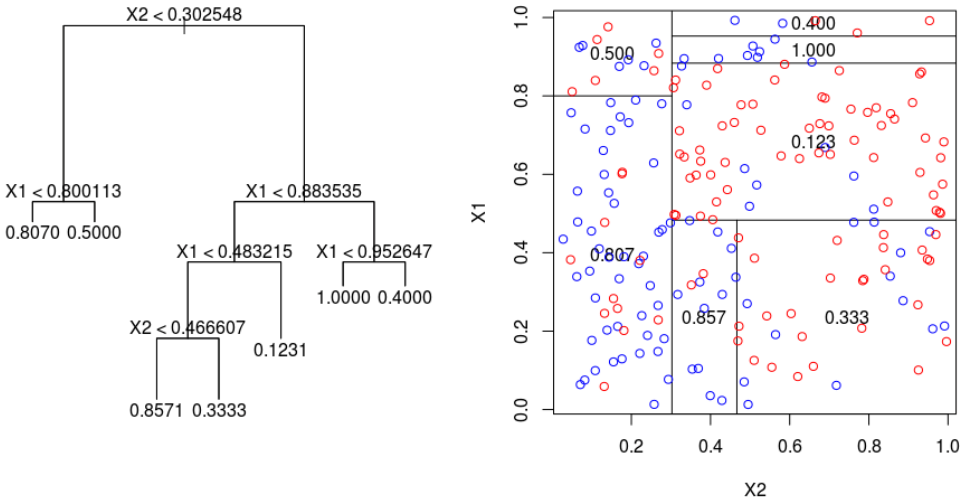


Figure 5.5: How decision tree regressor works. ([55])

1. **Builds n decision tree regressors (estimators).** The number of estimators n defaults to 100 in Scikit Learn (the machine learning Python library), where it is called `n_estimators`. The trees are built following the specified hyperparameters (e.g. minimum number of samples at the leaf nodes, maximum depth that a tree can grow, etc).

2. **Average prediction across estimators.** Each decision tree regression predicts a number as an output for a given input. Random forest regression takes the average of those predictions as its final output.

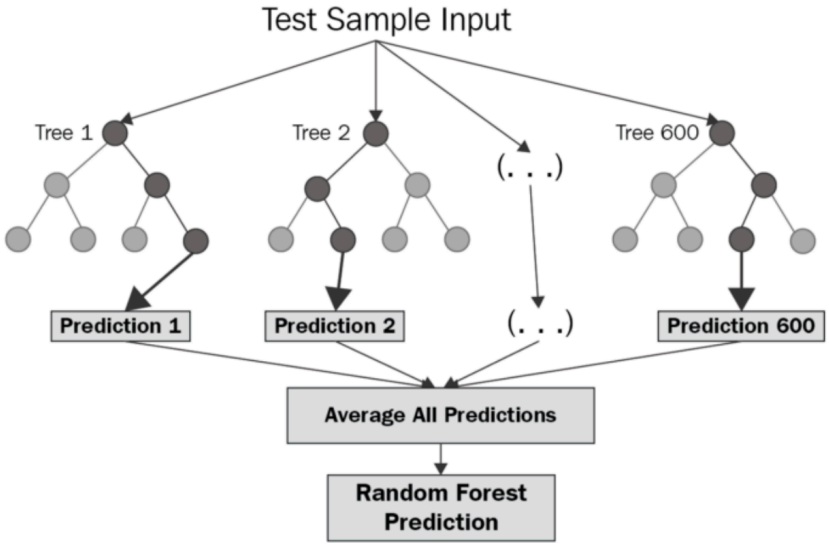


Figure 5.6: Random forest regression. ([58])

5.1.7 Multilayer Perceptron Regression

Multilayer Perceptrons (MLP) are the most classic example of deep learning networks. It usually consist of an input layer where we have the nodes which take as

values our data/features. Then we have some hidden nodes and an output layer. In Fig. 5.7 we can see a network with one input, one hidden and one output layer. Of course, we can put more that one hidden layer in our model, but it doesn't mean necessarily that a more complex model will lead us to better predictions.

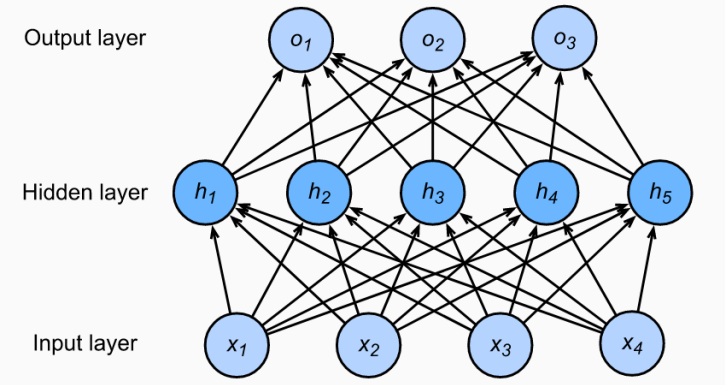


Figure 5.7: Multi Layer Perceptron. ([58])

When we use MLPs for classification we usually have a sequence of nodes in output layer and the equations we use for the forward process are,

$$\boldsymbol{h} = \boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1 \tag{5.15}$$

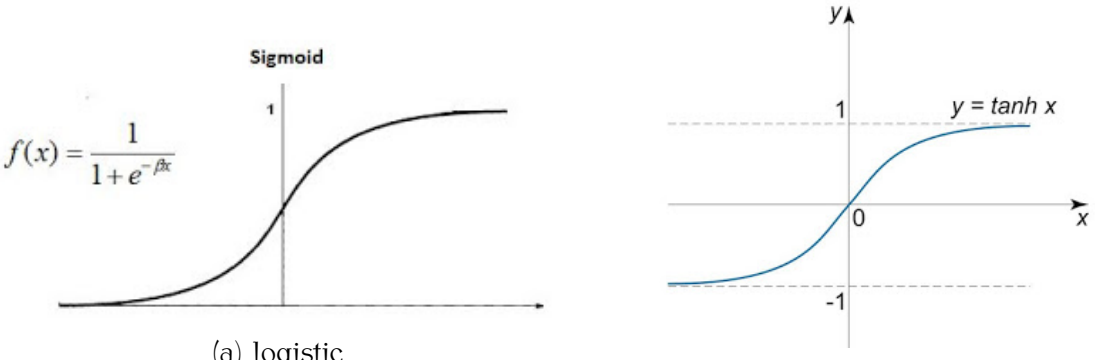
$$\boldsymbol{o} = \boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2 \tag{5.16}$$

$$\hat{\boldsymbol{y}} = \phi(\boldsymbol{o}) \tag{5.17}$$

where ϕ is a non linear function. And we can write alternatively,

$$\boldsymbol{o} = \boldsymbol{W}_2\boldsymbol{h} + \boldsymbol{b}_2 = \boldsymbol{W}_2(\boldsymbol{W}_1\boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2 = (\boldsymbol{W}_2\boldsymbol{W}_1)\boldsymbol{x} + (\boldsymbol{W}_2\boldsymbol{b}_1 + \boldsymbol{b}_2) = \boldsymbol{W}\boldsymbol{x} + \boldsymbol{b} \tag{5.18}$$

where $\boldsymbol{W} = \boldsymbol{W}_2\boldsymbol{W}_1$ and $\boldsymbol{b} = \boldsymbol{W}_2\boldsymbol{b}_1 + \boldsymbol{b}_2$. Usually for non linear function ϕ we use either a logistic function or an hyperbolic tangent, which are sigmoid functions.



(a) logistic

(b) tanh

Figure 5.8: Non-linear functions for MLPs

According to [57], the training process of the MLP occurs by continuous adjustment of the weights of the connections after each processing. This adjustment is based on the error in output (which is the difference between the expected result and the output). This continuous adjustment of the weights is a supervised learning process called backpropagation. The backpropagation algorithm consists of two parts: (a)

forward pass, (b) backward pass. In the forward pass, the expect output corresponding to the given inputs are evaluated. In the backward pass, partial derivatives of the cost function with respects to the different parameters are propagated back through the network. In case of regression, we usually have one output node which predicts the response variable $\boldsymbol{y}$.

5.2 Machine Learning on the Non-Critical Region

In this section we train the regressors we need to reconstruct non-critical region (low and high temperatures). The only difference between this case and critical region is that here we do a kind of pre-processing with some similarity measures. The reason why we do that is because in low and high temperatures there are no intense fluctuations and thus we need high accuracy.

5.2.1 Data

In this thesis we have two simulations: Metropolis-Hastings and Gaussian surfaces, and we want to find a connection between them. Ideally, we want to find the parameters with which we will feed Gaussian surfaces simulation to produce an Ising lattice in a specific temperature.

Consequently, to create data for each simulation we must do two things,

- We run Metropolis-Hastings algorithm for 600 temperatures in the interval $T \in [1.5, 4]$. We assume lattices of size $(64, 64)$.
- We create a dataset doing grid search in Gaussian Surfaces simulation with parameters:
  - Lattice $(64, 64)$ with fixed $rms = 3$ and $a = 0.6$.
  - Threshold[1] in range $[0, 7]$ with step 0.025.
  - Correlation length in range $[0.5, 5]$ with step 0.5.

Similarity Measures: For each one of the configurations we compute two similarity measures: magnetization $\langle M \rangle$ and VoM. Now the question is: *what is VoM? VoM* stands for 'Variance of Means' or 'Variance of Magnetizations'. To compute it we split each dimension of each lattice in 8 equal intervals. Having done that, we have split our lattices in 64 squares. For each square we compute the mean (magnetization) and then we compute the variance of means of all boxes. This is a similarity measure that help us to restrict correlation lengths, whereas mean magnetization help us to restrict threshold[2].

[1] Note that we chose only positive thresholds. This is because we want to learn only one sign of mean magnetization. If we don't do that, we can see from Fig. 1.2 that due to phase transition that occurs in the critical point, mean magnetization is not a function of temperature for low temperatures. The change of threshold causes changes in magnetization, and if we want to use Machine Learning algorithms with high accuracy we should choose sign for mean magnetization $\langle M \rangle$ and threshold t.

[2] We chose only threshold and correlation length to vary. This is because of two reasons: (1) we want to keep our problem as simpler as possible to have a clear picture, (2) the parameter rms doesn't add any extra information and this is because it is a three dimensional variance (however we work in two dimensions).

At this point we have two datasets: one dataset from Metropolis-Hastings and one dataset from Gaussian Surfaces. The main issue is that Gaussian Surfaces dataset has no temperature because it describes a static phenomenon. Therefore, we can compare the two simulation data and correspond a temperature to the Gaussian simulation data in respect to similarity measures and then use Machine Learning.

5.2.2 Corresponding temperatures to Gaussian simulation data with similarity measures

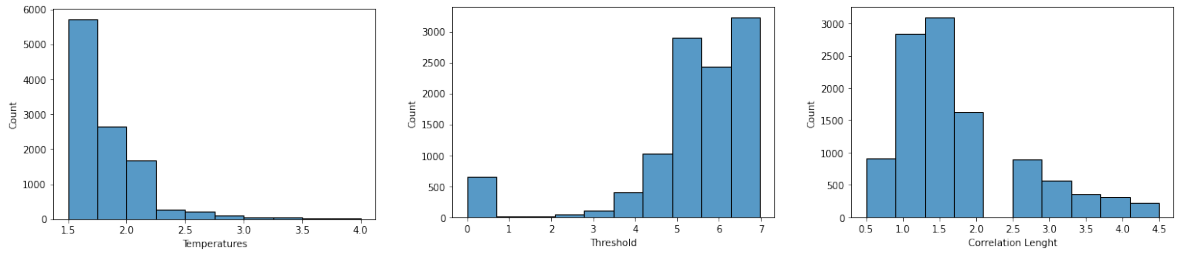
For each temperature of Metropolis-Hastings simulation we have computed two similarity measures $\langle M \rangle_{Ising}$ and VoM_{Ising}. Therefore, we run a loop over the 600 temperatures of Metropolis-Hastings data, and for each temperature we search which of the $\langle M \rangle_{Gauss}$ and VoM_{Gauss} are close to those of Metropolis-Hastings algorithm in respect of some tolerances $\epsilon_{\langle M \rangle}$ and ϵ_{VoM}.

Therefore, our criterion is

$$|\langle M \rangle_{Ising} - \langle M \rangle_{Gauss}| < \epsilon_{\langle M \rangle} \tag{5.19}$$

$$|VoM_{Ising} - VoM_{Gauss}| < \epsilon_{\text{VoM}} \tag{5.20}$$

and we set $\epsilon_{\langle M \rangle} = 0.005$ and $\epsilon_{VoM} = 0.001$. After corresponding a temperature to each lattice of Gaussian surfaces simulation that it is possible, we keep only the lattices that succeeded to be corresponded to a specific Ising temperature.



(a) Temperature (b) Threshold (c) Correlation Length

Figure 5.9: Histograms of the parameters of interest.

Having done this kind of elimination, we can draw the histograms of each parameter of interest (Fig. 5.9). It is clear, that it is easier to find lattices in low temperatures than lattices of high temperatures. We have a problem with imbalanced data and as we saw in the previous chapter, this is a factor of complexity. To get rid off this problem, we propose to do a data cleaning and remove some unnecessary correlation lengths of low temperatures.

5.2.3 Data Cleaning

Generally, the term '*data cleaning*' is used for the process of removing outliers, infinite or nan values, or unnecessary data. In this thesis we can't have outliers or even infinite values due to the fact that we work with simulation data. However, it is possible to have some unnecessary data. We need to find at least one parametric path that can give us similar lattices to those of Ising model, but probably this elimination we did in the previous section may give us more than one parametric paths. In these

terms we should keep our attention only in one path of interest and discard all the other ones.

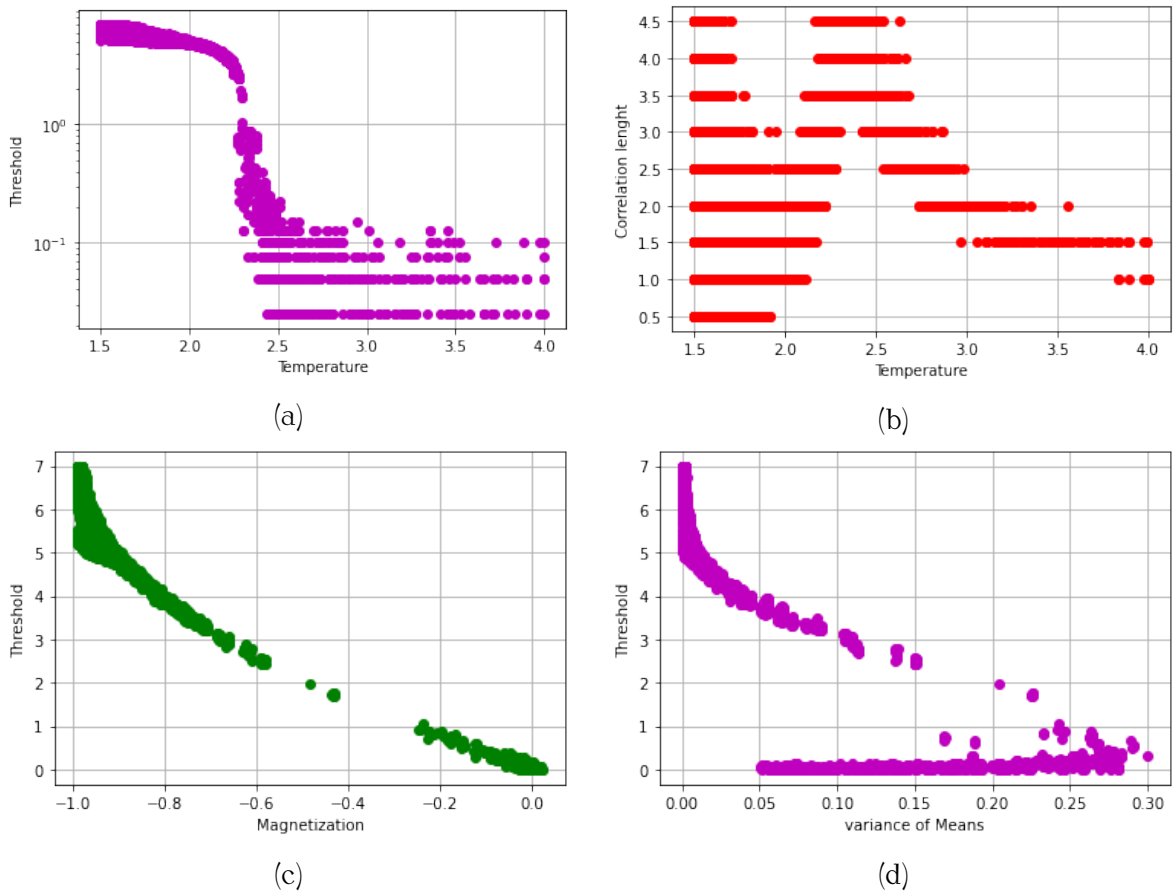


Figure 5.10: Some pair diagrams of temperature T, threshold t, correlation length ξ, mean magnetization $\langle M \rangle$ and VoM.

To have a clear picture of what happens with our parameters of interest, we have created these plots of Fig. 5.10. It is clear, that we have a relatively good behavior, and our problem seems to be learnable. Nevertheless, there is only one thing we should change. This is the strange behavior of correlation length ξ for low temperatures T (Fig. 5.10b), which there is not a good functional behavior because we see that for the same temperature we can choose any correlation length we want. This is not necessarily bad, but we want to help our algorithms to learn easily our data to have better accuracy.

To fix this issue, we move our hands and remove all the data for which $T < 1.9$ and $\xi > 1$. This is a completely arbitrary decision, since we could remove all the data for which $T < 1.9$ and $\xi > 1.5$ (or $\xi > 2$). However, we have the right to take this kind of arbitrary decisions, since we need to keep at least one parametric path. In Fig. 5.11 we can see the same diagrams after removing the unnecessary values of correlation length. Now both diagrams in Fig. 5.11a and 5.10b have a functional behavior in respect to temperature, and thus it makes sense to train regression models with these data.

A more convenient and easy way to illustrate the relation between correlation length and threshold is to create a contour plot. In Fig. 5.12 we can see that for low temperatures we need big threshold and for high temperatures we need small threshold and correlation length. To reconstruct the critical region we need big

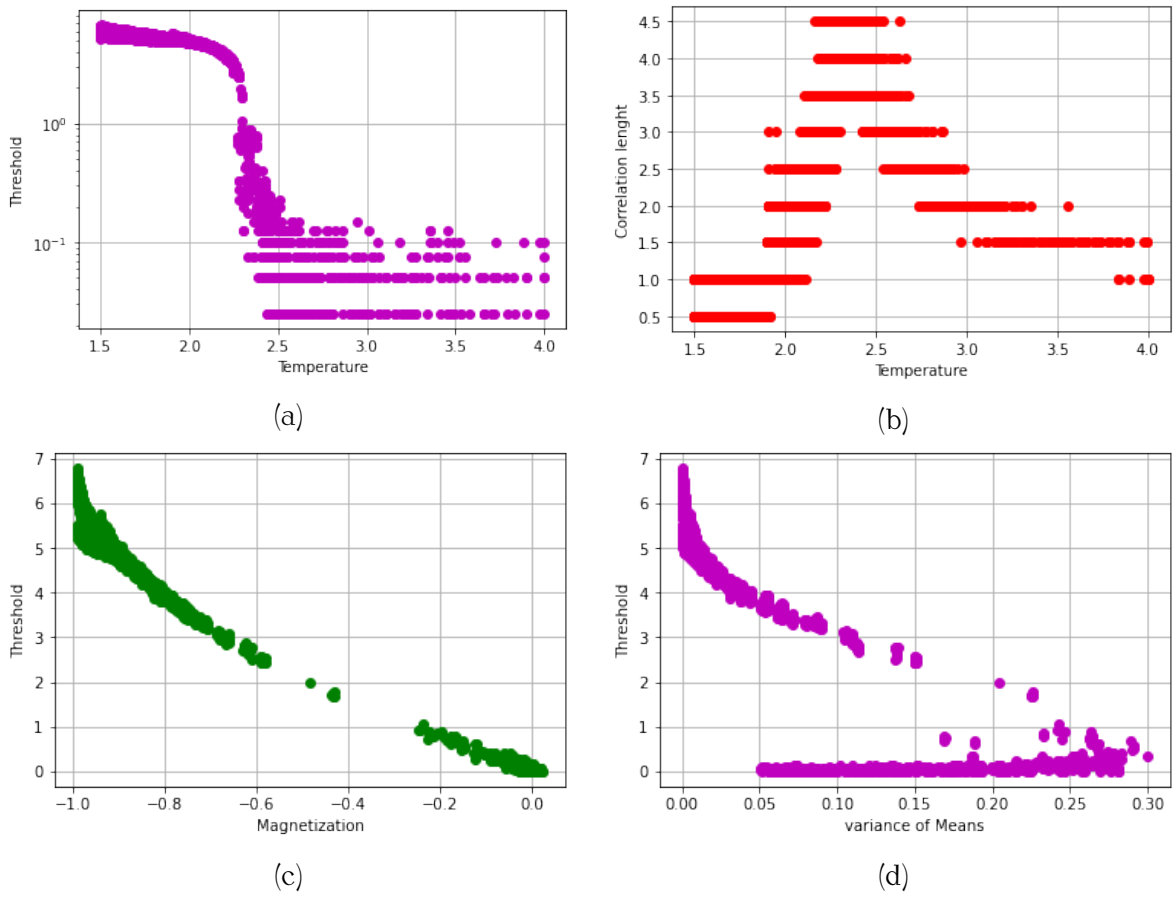


Figure 5.11: Some pair diagrams of temperature T, threshold t, correlation length ξ, mean magnetization $\langle M \rangle$ and VoM (after data cleaning).

correlation length and small threshold. This color map is a very important result of our work, because for a given threshold t and correlation length ξ, we can extract a lattice in a specific temperature T. However, this contour plot has a very important disadvantage: the parameters t and ξ we have are in a discrete interval, and we want to produce a lattice for any temperature. This is the reason why we need to introduce Machine Learning to our work.

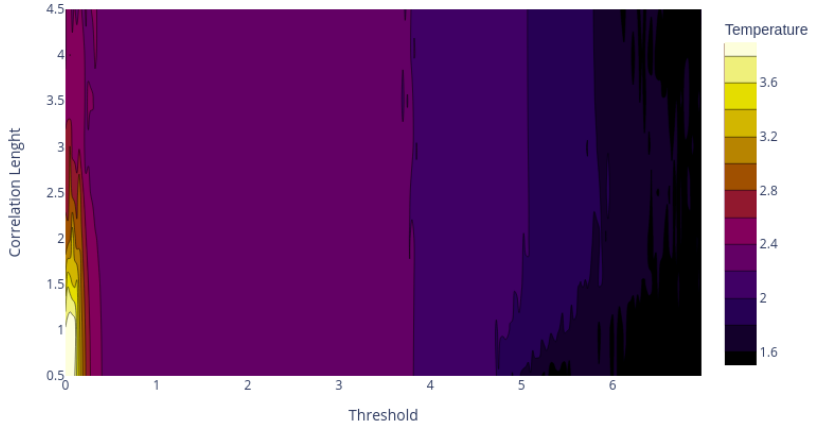


Figure 5.12: A contour plot of correlation length versus threshold. Different temperatures with different colors.

5.2.4 Regression

Having done the previous steps, we can train our models. Our main purpose is to find how threshold t and correlation length ξ depend on temperature T, mean magnetization $\langle M \rangle$ and VoM. We also split our data in train and test in proportion 70% train and 30% test. Thus we need to find a function,

$$[t, \xi] = \boldsymbol{G}_{\text{non-critical}}(T, \langle M \rangle, VoM) \tag{5.21}$$

and this is of course another multi-regression problem. Because of all this elimination we did, we see in Tab. 5.1 that even classic regression algorithms are able to do good predictions. We have a very good score for polynomial regression (with 94.24% for training set and 94.02% for test set) and Nearest Neighbors regression (with 95.09% for training set and 93.83% for test set). Random forest and MLP offers also good results. For MLP we use the default network of `scikit-learn` python library.

Algorithm	Train Score	Test Score
Linear Regression	88.96%	88.47%
Polynomial Regression (deg= 4)	94.25%	94.02%
Nearest Neighbors (nn= 8)	95.09%	93.83%
SVM (linear kernel)	87.27%	86.44%
SVM (poly kernel)	89.99%	89.80%
SVM (rbf kernel)	90.70%	90.61%
Decision tree	99.49%	88.70%
Random Forest	98.63%	92.78%
MLP	91.64%	91.57%

Table 5.1: Regression Results for Machine Learning different algorithms for the non-critical region.

5.3 Machine Learning on the Critical Region

We know that in critical region there are very intense fluctuations in magnetization. This is an important feature of Metropolis-Hastings algorithm and we need it in our reconstruction algorithm. Therefore, we need a kind of flexibility in critical region and the only way to ensure this property is by manipulating our data appropriately.

5.3.1 Data

We create a dataset doing grid search in Gaussian Surfaces Simulation with parameters:

- Lattice $(64, 64)$ with fixed $rms = 3$ and $a = 0.6$.
- Threshold in range $[-7, 7]$ with step 0.025.

- Correlation length in range $[0.5, 5]$ with step 0.5.

Note that here we learn the data of Gaussian surfaces simulation without using Metropolis-Hastings. Moreover, we choose both positive and negative values for our threshold, this is because we want to reassure that fluctuations won't be bounded on zero.

5.3.2 Regression

We said that in critical region we need flexibility. Therefore, the only thing we do is to learn how threshold t and correlation length ξ depend on mean magnetization $\langle M \rangle_{Gauss}$ and VoM_{Gauss} without using the dataset we produced with Metropolis-Hastings algorithm. Therefore, we search for a function,

$$[t, \xi] = [f_{\text{critical}}(\langle M \rangle, VoM), g_{\text{critical}}(\langle M \rangle, VoM)] = \boldsymbol{G}_{\text{critical}}(\langle M \rangle, VoM) \tag{5.22}$$

Consequently what we do is to split our dataset in train (70%) and test (30%) and we fit different models to see which match with our data better. As we can see from the Tab. 5.2, the best algorithms are the non-linear ones such as kNN, Decision Tree, Random Forest or MLP[3]. For the Multilayer Perceptron algorithm we used the de-

Algorithm	Train Score	Test Score
Linear Regression	62.98%	63.11%
Polynomial Regression (deg= 5)	87.01%	87.16%
Nearest Neighbors (nn= 20)	94.04%	93.47%
SVM (linear kernel)	61.00%	61.34%
SVM (poly kernel)	71.29%	71.52%
SVM (rbf kernel)	85.02%	85.26%
Decision tree	98.93%	88.22%
Random Forest	98.15%	92.33%
MLP	92.58%	92.62%

Table 5.2: Regression Results for Machine Learning different algorithms for the critical region.

fault configuration of `skicit-learn` python package. At this point we understand why Machine Learning is useful to our problem: because for critical region we have better results with non-classical regressors. Perhaps, somebody can do a grid search to obtain better scores. However, this is something that it won't increase the total efficiency of the algorithm very much. In our problem there are more important factors, like the similarity measures we choose for the critical region, or which parameters we will learn, or when we must choose a specific sign of magnetization. This is a problem that needs physical intuition, and thus our main aim is not to construct a Machine Learning algorithm with 100% scores, but to obtain results with physical meaning.

[3]We have two features and two variables which we want to predict. Therefore, we need to use multiregression and not just regression.

The main result of this algorithm is that we can predict a value for the correlation length ξ and threshold t given some values for the mean magnetization $\langle M \rangle_{\text{Gauss}}$ and VoM_{Gauss}. Here we don't care about temperature, and this is due to the fact that in critical region the probability distribution of mean magnetization flattens and thus we don't need high accuracy.

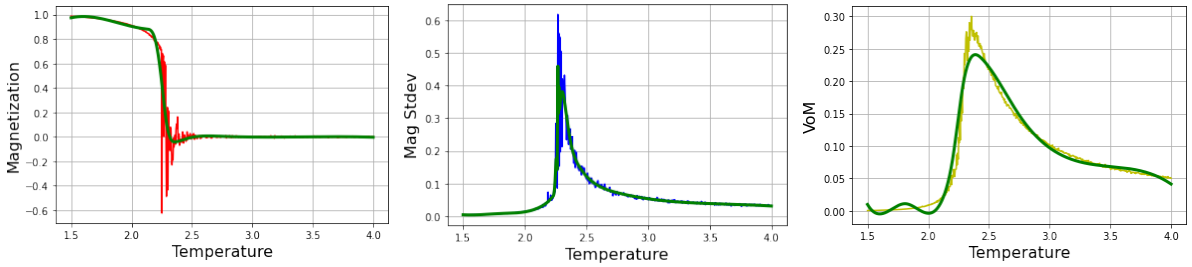
5.4 The final algorithm

At this point, it looks like we have all the tools we need to reconstruct Ising model with Gaussian surfaces simulation. The only thing we need to do is to put all the pieces of the puzzle in the correct sequence, and we will have an algorithm that will be able to reconstruct Ising model.

5.4.1 Splines interpolation

Our purpose is to create an algorithm that given a temperature T, will be able to give an Ising lattice, but instead of Metropolis-Hastings, it will use Gaussian Rough Surfaces simulation. So we can split this process into two steps: (1) for a given temperature, give some values for $\langle M \rangle$, $\sigma_{\langle M \rangle}$ and VoM via interpolation[4], and (2) given these parameters, predict the parameters of Gaussian surfaces simulation.

To do the first step, we use splines interpolation technique. For the only thing we should take care is the smoothing factor of splines interpolation. For magnetization we choose a smoothing factor $s = 3$, for standard deviation of magnetization $s = 0.3$ and for VoM, $s = 0.1$.



(a) Magnetization (b) StDev of Magnetization (c) VoM

Figure 5.13: Splines interpolation in Metropolis-Hastings data.

5.4.2 Algorithm's flowchart

There is not any better way to understand how our algorithm work from a flowchart. In Fig. 5.14 we can see the steps we must follow to produce lattices from Gaussian surfaces simulation that really look like Ising lattices.

So let's try to describe and understand this flowchart! As we said before, our input is a temperature T. Then we use the splines we created to interpolate the data of Metropolis-Hastings algorithm, and we take some estimation of the values $\langle M \rangle$, $\sigma_{\langle M \rangle}$ and VoM. Having these estimations, we can give some freedom to our

[4]This standard deviation of magnetization $\sigma_{\langle M \rangle}$ is the standard deviation of the distribution of magnetization over the equilibrium time/mc steps of Metropolis-Hastings simulation.

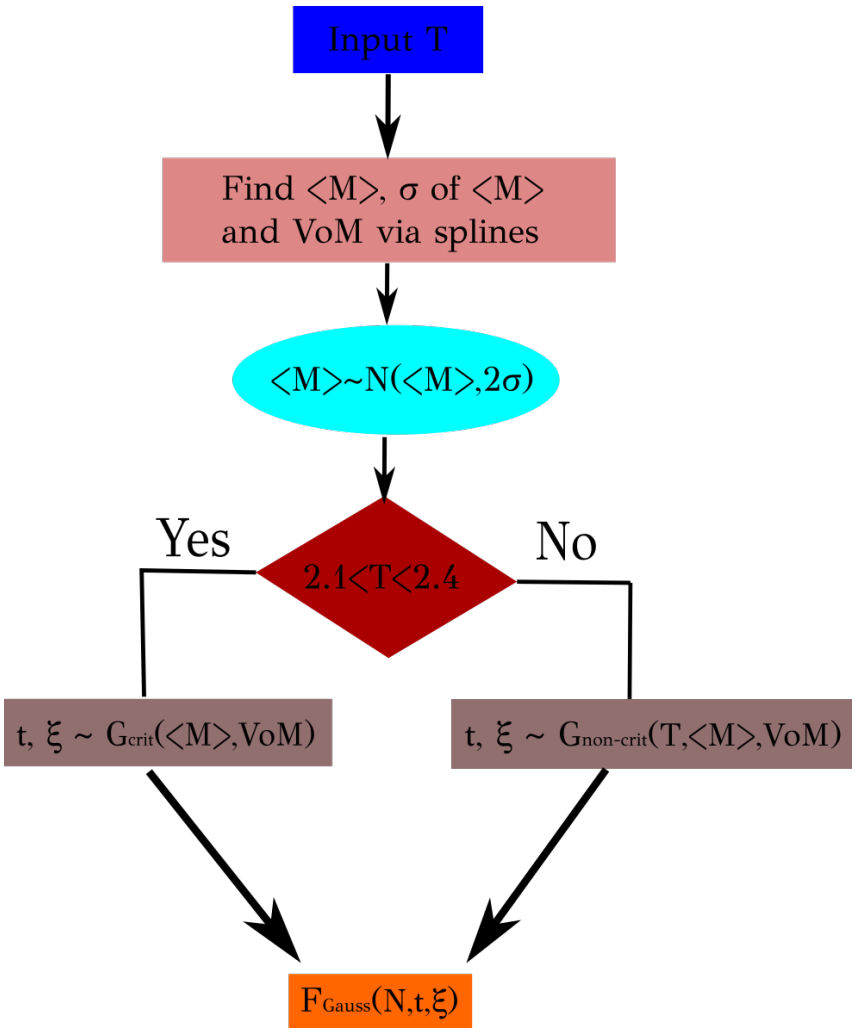


Figure 5.14: The flowchart of the algorithm which reconstruct Ising model via Gaussian surfaces simulation.

mean magnetization, by simulating it from a normal distribution with mean $\langle M \rangle$ and standard deviation $2\sigma_{\langle M \rangle}$. This is an important move to reconstruct the fluctuations of Ising model. If we want the probability distribution of mean magnetization to be the same as Metropolis-Hasting's, we must do this step to reassure that we have the same quantitative behavior of fluctuations. Then we simulate from our Machine Learning predictors G_{critical} and $G_{\text{non-critical}}$ depending on the temperature we chose. For G_{critical} and $G_{\text{non-critical}}$ we can choose any regressor we like. In this case, we opt random forest for critical region and MLP regressor for non-critical region. At the end of the day, we end up with two parameters: threshold t and correlation length ξ, which we use to simulate a lattice via Gaussian surfaces simulation F_{Gauss}.

Note that this flowchart stands for one temperature. If we really want to reconstruct Ising model, we must simulate for different temperatures, and repeat this loop for each one of them (to reconstruct fluctuations).

5.5 Diagnostics and Results

5.5.1 The parametric path

Let's take 160 different temperatures in the interval $T \in [1.5, 3.5]$ and do 1000 repeats for each temperature. For each temperature we can compute the average threshold and correlation length (over the 1000 repeats that we do). In this manner, we can plot the parametric path we should follow in Gaussian Surfaces simulation to have similar results as Ising model's (Fig. 5.15). Low temperatures are in the upper left corner, and as we move towards higher temperatures, we decrease our threshold until we reach the horizontal line of $t = 0$. For critical region temperatures the line is statistically perpendicular to correlation length axis.

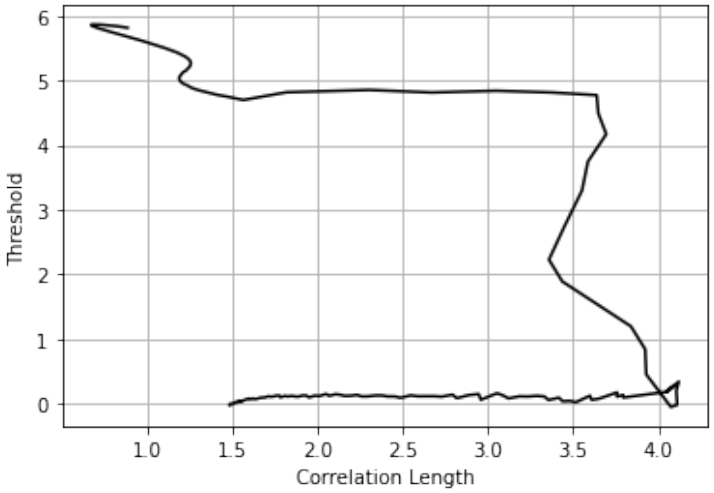


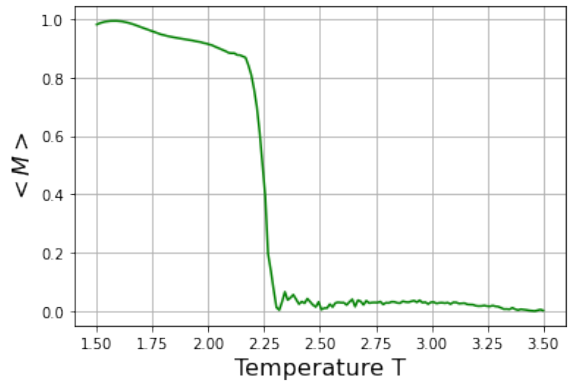
Figure 5.15: The parametric path we should follow in Gaussian surfaces simulation to have similar results as Ising model's.

5.5.2 Complexity Measures

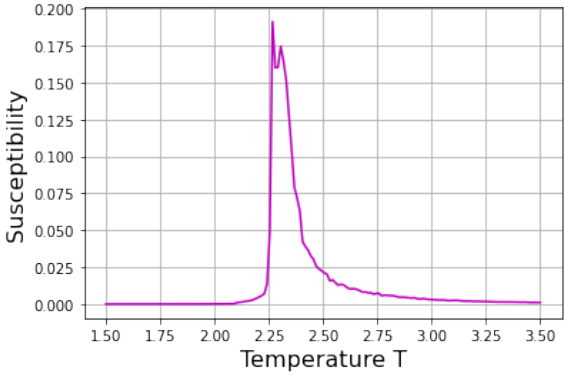
The next question we must answer is about the efficiency of our algorithm. Perhaps we succeeded to produce a parametric path, but is it really something similar to Ising model? In Fig. 5.16 we can see some diagnostics of our algorithm. These are some statistical quantities that we also computed for Ising model. Our algorithm looks very capable in reconstructing magnetization 5.16a. This is something that we expected because we based our algorithm on this statistical measure. Moreover, in Fig. 5.16b we can see that we succeeded to have the same qualitative behavior of susceptibility. Finally, we plotted some complexity measures in Fig. 5.16c, 5.16d and 5.16e and we see that all of them can detect the criticality. All these diagnostics show us a positive picture for our algorithm.

5.5.3 Correlations and probability distributions

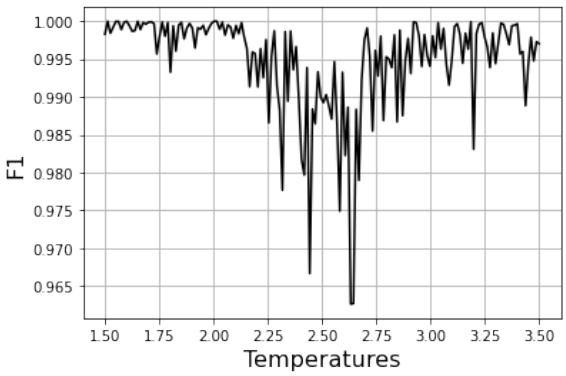
Another interesting plot is this of cross correlations in Fig. 5.17. To compute cross correlations we start from a particular spin/pixel of our lattice and we compute the product $s_i s_{i\pm r}$ where r is a distance from the chosen spin/pixel. We repeat this process for each spin/pixel of our lattice and we take the average in respect to spins



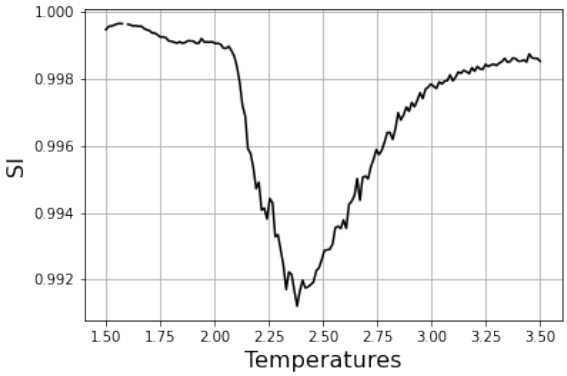
(a) Magnetization



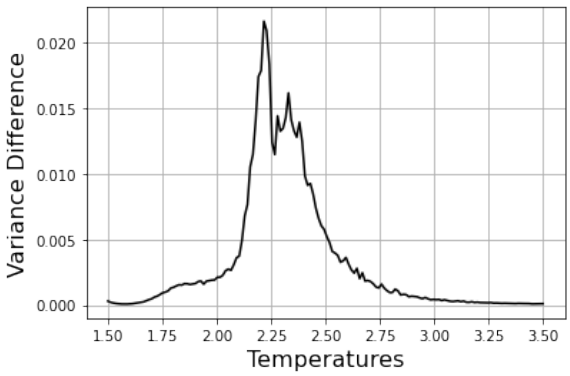
(b) Susceptibility



(c) F1



(d) Seb index



(e) DoV

Figure 5.16: The diagnostic of Gaussian surfaces simulation.

s_i for each r. Then we subtract the square of magnetization $\langle M \rangle^2$ and create a plot of cross correlations versus r. We expect in critical point to have a power law behavior, due to the fractality. In our diagram we have taken the logarithm of x and y axis, and thus it satisfies this observation.

Furthermore, in Fig. 5.18 there is the probability distribution of magnetization that we extracted from our Gaussian simulation algorithm for three different temperatures. We have the expected behavior, since the probability distribution is normal for non-critical region and it flattens for critical region. In Fig. 5.19 we can also see the autocorrelations of magnetization. It is clear that our samples are completely uncorrelated. This is an important pros of our algorithm since it means that we don't need to take a sampling rate to have uncorrelated results.

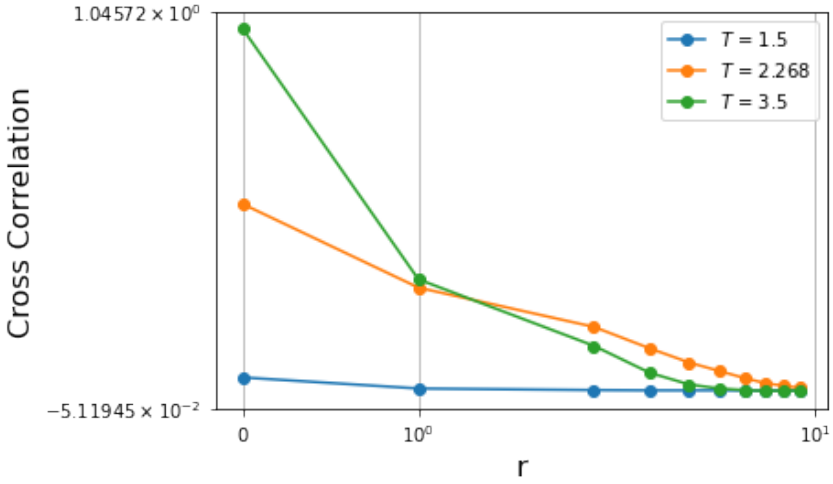


Figure 5.17: Cross correlations from Gaussian surfaces algorithm.

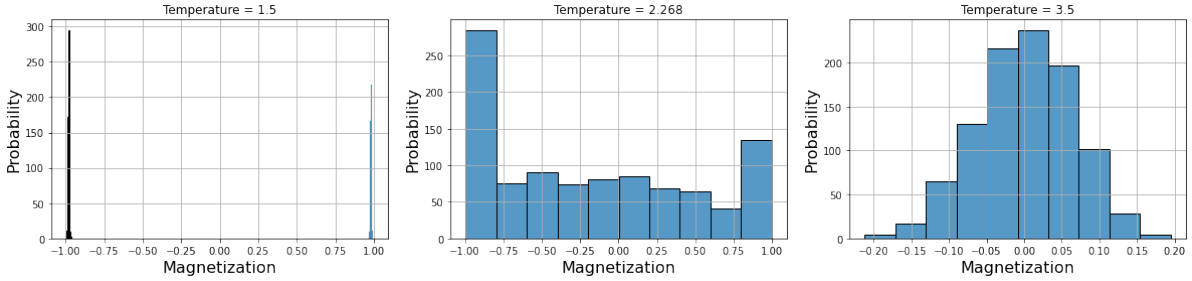


Figure 5.18: Probability distribution of magnetization from Gaussian surfaces algorithm.

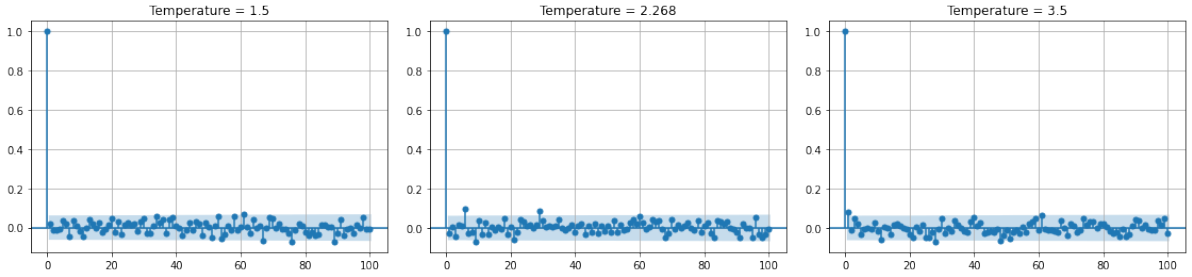


Figure 5.19: Autocorrelations of magnetization from Gaussian surfaces algorithm.

5.6 Conclusions

From this section, we understand that there is a connection between Gaussian surfaces simulation and Ising model. We constructed an algorithm that can reconstruct both Ising lattices and statistical characteristic of Ising model in a relatively good way. However, there is still a question about if this algorithm can substitute Metropolis-Hastings algorithm. Let's sum up the advantages of our algorithm,

- Gaussian surfaces don't need any Markov process and therefore, we don't need to wait until our system reaches the equilibrium.
- Gaussian surfaces simulation is very fast. If we want to reconstruct the whole range of 160 temperatures in the interval $T \in [1.5, 3.5]$ the time we need is

relatively the same as this of Metropolis-Hastings algorithm. However, this is not a fair comparison[5] since for Gaussian surfaces simulation we used `numba` accelerator, whereas for Gaussian surfaces we didn't use anything like that.

- The samples are uncorrelated and thus we don't need to use any sampling rate.
- Alternatively, one can still use Metropolis-Hastings algorithm and feed a Gaussian surface lattice as an initial condition to reach faster the equilibrium.

And these are only the computational advantages of the algorithm because apart from them, there are theoretical implications of our results. For instance, our approach give us a connection between two different algorithms. In this manner, we can see Ising lattices as a parametric path of some Gaussian Rough Surfaces, and vice versa Gaussian Surfaces simulation as a system for which, if we choose a particular parameter path, we can create phase transitions. Nevertheless, there is still an open question about if our algorithm can really produce all the microstates of Ising model. This question may have not an obvious answer, but we can assume that if there are some microstates that cannot be produced, these are the microstates with very small probabilities. However, this is just a conjecture.

[5]If we want to do a quantitative comparison, we need about 53.8 ms to produce a lattice with Gaussian Surfaces simulation, whereas we need about 7.97 s to run a Metropolis-Hastings algorithm for 10^8 Monte Carlo steps and sampling frequency $f = 5 \cdot 10^5$. With this sampling frequency we can produce 200 lattices, which means that the time to produce 200 lattices with Gaussian Surfaces simulation is about 10.1 s. However, this is not a fair comparison because we have used `numba` for Metropolis-Hastings acceleration.

We shall not cease from exploration, and the end of all our exploring will be to arrive where we started and know the place for the first time.

- T.S. Eliot

Epilogue

In this thesis we saw that it is possible to create a mapping between Rough Surfaces simulation and Ising model. To do that, we defined some complexity measures, which used them as diagnostics of this algorithm. These complexity measures gave us a new theoretical aspect about the criticality of Ising model. Particularly, their global minimum were connected with the aggregates that are shaped in critical temperature due to the high fractality. Consequently, we saw that $F1$ complexity measure (which has a completely geometrical interpretation) can detect the criticality, and we also defined some other measures that were inspired from Boltzmann Machines. To define them we can either split one lattice in two parts or compare two different lattices. We did a short discussion about the necessity to compute these measures in Boltzmann Machines, but at this point we just used them as diagnostics for our Machine Learning algorithm.

To create this Machine Learning algorithm that simulates Ising model with Gaussian Rough Surfaces simulation, we split our problem in two parts: critical temperatures and non-critical temperatures. For critical temperatures we used both positive and negative thresholds, whereas for non-critical temperatures we used only positive thresholds and compared Gaussian surfaces lattices with Metropolis-Hastings lattices in respect to two similarity measures: mean magnetization and VoM. The reason why we did that was because in critical region we need to have intense fluctuations in mean magnetization, whereas in non-critical region we need to have more accurate results. We used Machine Learning algorithms to find a function that connects threshold and correlation length of Rough Surfaces simulation with the magnetization and VoM of Ising model. This work can be done with classical regression algorithms also, however Machine Learning algorithms are more precise and can give us more accurate results.

Our main result was the parametric path that we should follow in Gaussian Surfaces simulation to produce a sequence of lattices that looks like Ising model. From a theoretical scope, we found a connection between two completely different fields. And this can have two interpretations: the first and most important one is that it is possible to find an alternative way to simulate Ising model, and the second one is that we can choose a parametric path in Gaussian Surfaces so as to create a dynamical system with a phase transition. If we focus our attention in the first interpretation, we saw that a Gaussian Surfaces simulation can produces uncorrelated samples that are already in equilibrium state without running any Markov chain simulation. In these terms, we can slide over the two main disadvantages of Metropolis-Hastings algorithm. Finally, we computed some of the statistical and complexity measures of Ising model, and we saw very high similarity between the two representations.

Part III

Appendix

"Talk is cheap. Show me the code."

- Linus Torvalds

A Algorithms

A.1 Rough Surfaces Simulation

This function can simulate either Gaussian or non-Gaussian surfaces (`non_Gauss=False` or `True` respectively). We want to remind that there was a problem with the non-Gaussian translator system and the algorithm didn't converge. Also there is another one property of the algorithm: for `off=True` it doesn't plot the visualizations, whereas for `off=False` it plots them.

```
def SAimage_fft_2(N_total=500,rms=3,skewness=1,kurtosis=3,corlength_x=10,corlength_y=10,alpha=0.9,th=1000.0,non_Gauss=True
  ,z_axis=[-2000,5000],off=False):
    '''
    Input:

    Generation of non-Gaussian surfaces with predetermined
    1. Mean value (mu)
    2. Standard deviation (rms)
    3. Skewness (skew)
    4. Kurtosis (kurt)
    5. Correlation lengths (ksix, ksiy)
    6. Roughness exponent (aplha)

    The input values of kurtosis and skewness should satisfy the inequality
    kurtosis > 1 + skewness^2
    The method is described in Yang et al. CMES, vol.103, no.4, pp.251-279,2014
    We use pearson translation to transform random noise to non-gaussian series and
    inverse Fourier transform to generate Gaussian surfaces.

    The method is implemented in three steps
    1. Generation of a Gaussian self-affine surface z_gs with rms, ksix,ksiy,alpha
    2. Generation of non-Gaussian noise z_ngn with mu,rms,skew,kurt
    3. Arranging z_ngn according to the spatial arrangement of z_g to get a non-Gaussian self-affine surface z_ngs
    '''

    # Input
    N = N_total
    numpoints = N
    hhcf_y = np.zeros(int(N/2))
    Pyy = 0.0

    # input moments and spatial parameters
    skew = skewness;
    kurt = kurtosis
    ksix = corlength_x
    ksiy = corlength_y
    alpha = alpha
    mu = 0

    # 1st step: Generation of a Gaussian surface

    # Determine the autocorrelation function R(tx,ty)
    R = np.zeros([numpoints+1,numpoints+1])
    txmin, txmax, tymin, tymax = -numpoints/2,numpoints/2,-numpoints/2,numpoints/2
    dtx, dty = (txmax-txmin)/(numpoints), (tymax-tymin)/(numpoints)

    tx = np.arange(txmin,txmax,dtx)
    ty = np.arange(tymin,tymax,dty)

```

```
    for txx in tx:
      for tyy in ty:
        R[int(txx+txmax+1),int(tyy+tymax+1)]=((rms**2)*np.exp(-(abs(np.sqrt((txx/ksix)**2+(tyy/ksiy)**2)))**(2*alpha)))

    # According to the Wiener-Khinchine theorem FR is the power spectrum of the desired profile
    FR = np.fft.fft2(R, s=[numpoints,numpoints])
    AMPR = np.sqrt(dtx**2+dty**2)*abs(FR)

    # 2nd step: Generate a white noise, normalize it and take its Fourier transform
    X = np.random.rand(numpoints,numpoints)
    aveX = X.mean(axis=0).mean(axis=0)
    dif2X = (X-aveX)**2;
    stdX = np.sqrt(dif2X.mean(axis=0).mean(axis=0));
    X = X/stdX;
    XF = np.fft.fft2(X, s=[numpoints,numpoints])

    # 3nd step: Multiply the two Fourier transforms
    YF = XF*np.sqrt(AMPR)

    # 4th step: Perform the inverse Fourier transform of YF and get the desired surface
    zaf = np.fft.ifft2(YF,s=[numpoints,numpoints])
    z = np.real(zaf)
    avez = z.mean(axis=0).mean(axis=0)
    dif2z = (z-avez)**2
    stdz = np.sqrt(dif2z.mean(axis=0).mean(axis=0))
    z = ((z-avez)*rms)/stdz

    # Define the fraction of the surface to be analysed
    xmin, xmax, ymin, ymax= 0, N, 0, N
    z_gs=z[xmin:xmax,ymin:ymax]
    Nh=xmax-xmin+1

    # For the Gaussian Surface Simulation
    if not non_Gauss:
      X = np.arange(0,N)
      Y = np.arange(0,N)
      X, Y = np.meshgrid(X, Y)

      if off==False:
        fig = plt.figure()
        fig.set_figwidth(8)
        fig.set_figheight(8)
        ax = plt.axes(projection='3d')
        ax.plot_surface(X,Y,z_gs,cmap='viridis', edgecolor='none')
        ax.view_init(30, 150)
        plt.title('Gaussian Surface',fontsize=16)
        plt.show()

        plt.imshow(z_gs<th, cmap='gray', origin='lower', interpolation='none')
        plt.title('Binarized Image from Gaussian Surface')
        plt.show()
      image = z_gs>th

    del R, AMPR, z

    # For the Non-Gaussian Surface Simulator
    if non_Gauss:
      # 2nd step: Generation of a non-Gaussian noise NxN

      # Finding the parameters of JohnsonSU distribution
      # Special thank to Max Pierini for this part of the code
      coef, j_type, err = f_johnson_M(mu, rms, skew, kurt)

      gamma, delta, xi, lam = coef

      ## Simulating from JohnsonSU distribution
      z_ngn = johnsonsu.rvs(a=gamma, b=delta, loc=xi, scale=lam,size=[N,N])

      #3rd step: Combination of z_gs with z_ngn to output a z_ngs
      v_gs = z_gs.reshape(-1)
      v_ngn = z_ngn.reshape(-1)
      Igs = np.argsort(v_gs)
      Ingn = np.argsort(v_ngn)
      vs_gs = np.sort(v_gs)
      vs_ngn = np.sort(v_ngn)

      v_ngs = np.zeros(np.max(Igs)+1)

      for iv in range(N*N):
        ivs = Igs[iv]
        v_ngs[ivs] = vs_ngn[iv]

      X = np.arange(0,N)
      Y = np.arange(0,N)
      X, Y = np.meshgrid(X, Y)
      z_ngs = -v_ngs.reshape(N,N)
      # Creating color map
      my_cmap = plt.get_cmap('Blues')


      if not off:
        fig = plt.figure()
        fig.set_figwidth(8)
        fig.set_figheight(8)
        ax = plt.axes(projection='3d')
        ax.set_xlim(0,N)
        ax.set_ylim(0,N)
        ax.set_zlim(z_axis)
        ax.plot_surface(X,Y,z_ngs,cmap=my_cmap, edgecolor='black')
        ax.view_init(30, 150)
```

```
149         plt.title(r'L={}, $\mu$={}, rms={}, skew={}, kurt={}, corx={}, cory={}, alpha={}, threshold={}'.format(N_total, mu,
    rms, skew, kurt, ksix, ksiy, alpha, th))
150         plt.suptitle('Non-Gaussian Surface',fontsize=20)
151         # Add a color bar which maps values to colors.
152         plt.show()
153
154         fig = plt.figure()
155         fig.set_figwidth(8)
156         fig.set_figheight(8)
157         plt.imshow(z_ngs>th, cmap='gray', origin='lower', interpolation='none')
158         plt.suptitle('Binarized Image from Non-Gaussian Surface',fontsize=20)
159         plt.title(r'L={}, $\mu$={}, rms={}, skew={}, kurt={}, corx={}, cory={}, alpha={}, threshold={}'.format(N_total, mu,
    rms, skew, kurt, ksix, ksiy, alpha, th))
160         plt.show()
161
162         # Correlation Functions
163         # a. 1-D height-height correlation function
164         inpsur=z_ngs;
165         hhcf1d=np.zeros(N//2)
166         rdif=np.zeros(N//2)
167         for ndif in range(N//2):
168           surf1=inpsur[0:N,0:(N-ndif)]
169           surf2=inpsur[0:N,ndif:N]
170           difsur2=(surf1-surf2)**2;
171           hhcf1d[ndif]=np.sqrt(difsur2.mean(axis=0).mean(axis=0));
172           rdif[ndif]=ndif
173
174
175         plt.loglog(rdif,hhcf1d)
176         plt.grid()
177         plt.xlabel('log(r(nm))')
178         plt.ylabel('log(G(r) (nm))')
179         plt.title('1-D height-height correlation function (non-Gaussian surface)')
180         plt.show()
181
182       image = z_ngs>th
183
184       return image
185
```

A.2 Complexity Measures

Some of the complexity measures we used, were from `https://pypi.org/project/data-complexity/` by Son Nguyen. However, in this thesis we computed some complexity-measures that didn't exist in this repository, and we also defined some new ones. Here we present only author's work.

A.2.1 Classification Complexity Measures

```
1   def F2(X,y):
2     minmaxf1 = min(max(X[y==0][:,0]),max(X[y==1][:,0]))
3     maxminf1 = max(min(X[y==0][:,0]),min(X[y==1][:,0]))
4     maxmaxf1 = max(max(X[y==0][:,0]),max(X[y==1][:,0]))
5     minminf1 = min(min(X[y==0][:,0]),min(X[y==1][:,0]))
6
7     minmaxf2 = min(max(X[y==0][:,1]),max(X[y==1][:,1]))
8     maxminf2 = max(min(X[y==0][:,1]),min(X[y==1][:,1]))
9     maxmaxf2 = max(max(X[y==0][:,1]),max(X[y==1][:,1]))
10    minminf2 = min(min(X[y==0][:,1]),min(X[y==1][:,1]))
11
12    F2 = max([0,minmaxf1-maxminf1])/(maxmaxf1-minminf1)*\
13    max([0,minmaxf2-maxminf2])/(maxmaxf2-minminf2)
14
15    return F2
16
17  def F3(X,y):
18    minmaxf1 = min(max(X[y==0][:,0]),max(X[y==1][:,0]))
19    maxminf1 = max(min(X[y==0][:,0]),min(X[y==1][:,0]))
20
21    minmaxf2 = min(max(X[y==0][:,1]),max(X[y==1][:,1]))
22    maxminf2 = max(min(X[y==0][:,1]),min(X[y==1][:,1]))
23
24    i1 = 0
25    i2 = 0
26    n1 = len(X[:,0])
27    n2 = len(X[:,1])
28
29    for j in range(n1):
30      if X[j,0]>maxminf1 and X[j,0]<minmaxf1:
31        i1+=1
32
33    for j in range(n2):
34      if X[j,1]>maxminf2 and X[j,1]<minmaxf2:
35        i2+=1
36
37    n = n1*n2
38    F3 = min([i1/n,i2/n])
```

```
    return F3

```

A.2.2 Boltzmann Machine Inspired Complexity Measures

Here the index 2 in the name of a function stands for the complexity measure that compares two different lattices. For instance, `mean_diff()` is the DoM computed for one lattice that we have split it in two parts, and `mean_diff2()` compares two lattices. Of course, this work can be done with only one function, however the author preferred this way.

```
import pandas as pd
import numpy as np
from scipy.stats import skew
from scipy.stats import kurtosis

def image_to_dataset(image):
  '''
  Takes a lattice image and translates it to dataset
  '''
  coord = []
  label = []

  for i in range(len(image[0,:])):
    for j in range(len(image[:,0])):
      coord.append([float(j),float(i)])
      label.append(image[j,i])

  coord = np.array(coord)
  label = np.array(label)
  return coord, label

def seb_index(img):

  global N

  mean_input = np.mean(img[:,0:N//2])
  mean_hidden = np.mean(img[:,N//2:N])
  var_input = np.var(img[:,0:N//2])
  var_hidden = np.var(img[:,N//2:N])

  num = (np.abs(mean_input)-np.abs(mean_hidden))**2/4
  den = 0.5*var_input+0.5*var_hidden

  r = num/den

  seb_idx = 1/(1+r)

  return seb_idx

def seb2_index(img, hidden):

  mean_input = np.mean(img)
  mean_hidden = np.mean(hidden)
  var_input = np.var(img)
  var_hidden = np.var(hidden)

  num = (np.abs(mean_input)-np.abs(mean_hidden))**2/4
  den = 0.5*var_input+0.5*var_hidden

  r = num/den

  seb2_idx = 1/(1+r)

  return seb2_idx

def mean_diff(img):
  global N
  mean_input = np.mean(img[:,0:N//2])
  mean_hidden = np.mean(img[:,N//2:N])

  r = (np.abs(mean_input)-np.abs(mean_hidden))**2

  return r

def mean_diff2(img,hidden):
  mean_input = np.mean(img)
  mean_hidden = np.mean(hidden)

  r = (np.abs(mean_input)-np.abs(mean_hidden))**2

  return r


def var_diff(img):
  global N
  var_input = np.var(img[:,0:N//2])
  var_hidden = np.var(img[:,N//2:N])

  r = (np.abs(var_input)-np.abs(var_hidden))**2

  return r

def var_diff2(img,hidden):
  var_input = np.var(img)
```

```
    var_hidden = np.var(hidden)

    r = (np.abs(var_input)-np.abs(var_hidden))**2

    return r

  def skew_diff(img):
    global N
    skew_input = skew(img[:,0:N//2].flatten())
    skew_hidden = skew(img[:,N//2:N].flatten())

    r = (np.abs(skew_input)-np.abs(skew_hidden))**2

    return r

  def skew_diff2(img,hidden):
    skew_input = skew(img.flatten())
    skew_hidden = skew(hidden.flatten())

    r = (np.abs(skew_input)-np.abs(skew_hidden))**2

    return r

  def kurt_diff(img):
    global N
    kurt_input = kurtosis(img[:,0:N//2].flatten())
    kurt_hidden = kurtosis(img[:,N//2:N].flatten())

    r = (np.abs(kurt_input)-np.abs(kurt_hidden))**2

    return r

  def kurt_diff2(img,hidden):
    kurt_input = kurtosis(img.flatten())
    kurt_hidden = kurtosis(hidden.flatten())

    r = (np.abs(kurt_input)-np.abs(kurt_hidden))**2

    return r

  def var_of_mean(image,boxes=8):
    N = len(image[:,0])
    mu_matrix = np.zeros([boxes,boxes])
    for i in range(boxes):
      for j in range(boxes):
        mu_matrix[i,j] = np.mean(image[i*N//boxes:(i+1)*N//boxes,j*N//boxes:(j+1)*N//boxes])
    return np.var(mu_matrix)

  def var_of_mean_diff(image,hidden,boxes=16):
    N = len(image[:,0])
    mu_matrix1 = np.zeros([boxes,boxes])
    for i in range(boxes):
      for j in range(boxes):
        mu_matrix1[i,j] = np.mean(image[i*N//boxes:(i+1)*N//boxes,j*N//boxes:(j+1)*N//boxes])

    vof1 = np.var(mu_matrix1)

    mu_matrix2 = np.zeros([boxes,boxes])
    for i in range(boxes):
      for j in range(boxes):
        mu_matrix2[i,j] = np.mean(hidden[i*N//boxes:(i+1)*N//boxes,j*N//boxes:(j+1)*N//boxes])
    vof2 = np.var(mu_matrix2)

    r = (np.abs(vof1)-np.abs(vof2))**2
    return r


  def kl_divergence(p, q):
    return np.sum(np.where(p != 0, p * np.log(p / q), 0))


  def find_prob(m):
    p = np.empty(2)
    p[0] = np.count_nonzero(m == 1)/m.size
    p[1] = np.count_nonzero(m == -1)/m.size
    return p

  def sum_nondiag(m):
    S = np.sum(m)-np.trace(m)
    return S

  def prod_nondiag_diff(img):
    global N
    left_prod = img[:,0:N//2].reshape(-1,1)*img[:,0:N//2].reshape(-1,1).T
    right_prod = img[:,N//2:N].reshape(-1,1)*img[:,N//2:N].reshape(-1,1).T

    left = sum_nondiag(left_prod)/2
    right = sum_nondiag(right_prod)/2

    r = (np.abs(left)-np.abs(right))**2

    return r

  def prod_nondiag_diff2(img,hidden):

    left_prod = img.reshape(-1,1)*img.reshape(-1,1).T
    right_prod = hidden.reshape(-1,1)*hidden.reshape(-1,1).T

    left = sum_nondiag(left_prod)/2
    right = sum_nondiag(right_prod)/2

```

```
    r = (np.abs(left)-np.abs(right))**2

    return r

  def isingen_diff(img):
    global N,J,B

    en_input = getE2(img[:,0:N//2],N,N//2,J,B)
    en_hidden = getE2(img[:,N//2:N],N,N//2,J,B)

    r = (np.abs(en_input)-np.abs(en_hidden))**2

    return r

  def isingen_diff2(img,hidden):
    global N,J,B

    en_input = getE(img,N,J,B)
    en_hidden = getE(hidden,N,J,B)

    r = (np.abs(en_input)-np.abs(en_hidden))**2

    return r

```

A.3 Reconstructing Ising model with Gaussian Surfaces Simulation

A.3.1 Comparing Similarity Measures

```
import pandas as pd
import scipy as sc

ising_data = pd.read_csv('/kaggle/input/ising-ml/ising_regression.csv', header=None)
col_names = ['Nan','Temperature', 'Energy','Magnetization','Mag Stdev','Ising Variance',
'Ising Skewness','Ising Kurtosis','Var of Mean','Heat Capacity','Susceptibility',
'F1','seb_idx','var diff','skew diff','kurt diff']

ising_data.columns = col_names
del ising_data['Nan'] # Discard Nan column
ising_data = ising_data.drop(0,axis=0) # Discard first row
ising_data = ising_data.astype(float)
ising_data

surf_simulation_data = pd.read_csv('/kaggle/input/ising-ml/gaussian.csv')
del surf_simulation_data['Unnamed: 0'] # Discard Nan column
surf_simulation_data

error_mag = 0.005
error_vom = 0.001

for i in range(len(ising_data)):
  target_mag = np.abs(ising_data['Magnetization'].values[i])
  target_vom = ising_data['Var of Mean'].values[i]
  surf_simulation_data.loc[((np.abs(np.abs(surf_simulation_data['Magnetization'])-target_mag)<error_mag) &
  (np.abs(np.abs(surf_simulation_data['Var of Mean'])-target_vom)<error_vom)),
  ['Temperature']] = ising_data['Temperature'].values[i]

surf_simulation_data = surf_simulation_data[surf_simulation_data['Temperature'].notna()]

```

A.3.2 Data Cleaning and Machine Learning for the Non-Critical Region

```
def compute_metrics(model, x_tr, y_tr, x_ts, y_ts):
train_predict = model.predict(x_tr)
test_predict = model.predict(x_ts)

#   (Accuracy)
train_accuracy = r2_score(y_tr, train_predict)
test_accuracy = r2_score(y_ts, test_predict)

#
train_MSE = mean_squared_error(y_tr, train_predict)
test_MSE = mean_squared_error(y_ts, test_predict)

print('    : {:.2%}'.format(
  train_accuracy))
  print('    : {:.2%}\n'.format(test_accuracy))
    print('     : {:.4f}'.format(
    train_MSE))
    print('     : {:.4f}'.format(test_MSE))

    return train_predict, test_predict, train_accuracy, test_accuracy, train_MSE, test_MSE

##### Gaussian #####
surf_simulation_data = surf_simulation_data.drop(surf_simulation_data[(surf_simulation_data['Temperature'] < 1.9) &
```

```
    (surf_simulation_data['cor_lenght'] > 1.0)].index)

    X = surf_simulation_data[['Temperature','Magnetization','Var of Mean']].values
    y = surf_simulation_data[['cor_lenght', 'threshold']].values

    # load and summarize the dataset
    from sklearn.model_selection import train_test_split

    # split into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

    print('Train', X_train.shape, y_train.shape)
    print('Test', X_test.shape, y_test.shape)


    lr = MultiOutputRegressor(LinearRegression())
    lr.fit(X_train, y_train)
    train_predict, test_predict, train_accuracy, test_accuracy, train_MSE, test_MSE = compute_metrics(lr, X_train, y_train,
      X_test, y_test)


    pf2 = PolynomialFeatures(degree=4)

    x_train2 = pf2.fit_transform(X_train)
    x_test2 = pf2.fit_transform(X_test)

    start_time = time.time()
    pol4 = MultiOutputRegressor(LinearRegression())
    pol4.fit(x_train2, y_train)

    train_predict, test_predict, train_accuracy, test_accuracy, train_MSE, test_MSE = compute_metrics(pol4, x_train2, y_train,
       x_test2, y_test)


    neigh8 = MultiOutputRegressor(KNeighborsRegressor(n_neighbors=8))
    neigh8.fit(X_train, y_train)
    _ = compute_metrics(neigh8, X_train, y_train, X_test, y_test)


    regr = MultiOutputRegressor(SVR(kernel='linear'))
    regr.fit(X_train, y_train)
    _ = compute_metrics(regr, X_train, y_train, X_test, y_test)


    regr = MultiOutputRegressor(SVR(kernel='poly'))
    regr.fit(X_train, y_train)
    _ = compute_metrics(regr, X_train, y_train, X_test, y_test)


    regr = MultiOutputRegressor(SVR(kernel='rbf'))
    regr.fit(X_train, y_train)
    _ = compute_metrics(regr, X_train, y_train, X_test, y_test)


    tree = MultiOutputRegressor(DecisionTreeRegressor(random_state=0))
    tree.fit(X_train, y_train)
    _ = compute_metrics(tree, X_train, y_train, X_test, y_test)


    forest = MultiOutputRegressor(RandomForestRegressor(random_state=0))
    forest.fit(X_train, y_train)
    _ = compute_metrics(forest, X_train, y_train, X_test, y_test)


    MLP = MLPRegressor().fit(X_train, y_train)
    MLP.fit(X_train, y_train)
    _ = compute_metrics(MLP, X_train, y_train, X_test, y_test)

```

A.3.3 Machine Learning for the Critical Region

```
    surf_simulation_data_crit = pd.read_csv('/kaggle/input/ising-ml/gaussian_critical.csv')
    surf_simulation_data_crit = surf_simulation_data_crit.drop(surf_simulation_data_crit[(surf_simulation_data_crit['Image Var
      '] == np.NaN)].index)
    surf_simulation_data_crit = surf_simulation_data_crit.drop(surf_simulation_data_crit[(surf_simulation_data_crit['
      Magnetization'] == np.NaN)].index)
    surf_simulation_data_crit

    ##### Learn Surface Simulation Data for Critical Region Simulation #####
    X = surf_simulation_data_crit[['Magnetization','Var of Mean']].values
    y = surf_simulation_data_crit[['cor_lenght','threshold']].values

    # split into train and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1)

    print('Train', X_train.shape, y_train.shape)
    print('Test', X_test.shape, y_test.shape)

    critical = MultiOutputRegressor(RandomForestRegressor(random_state=0))
    critical.fit(X_train, y_train)
    _ = compute_metrics(critical, X_train, y_train, X_test, y_test)

```

A.3.4 Splines Interpolation

```
    is_temp = ising_data['Temperature'].values
```

```
is_mag = ising_data['Magnetization'].values
is_stdev = ising_data['Mag Stdev'].values
is_vom = ising_data['Var of Mean'].values

from scipy.interpolate import UnivariateSpline
mag_spl = UnivariateSpline(is_temp, is_mag)
stdev_spl = UnivariateSpline(is_temp, is_stdev)
vom_spl = UnivariateSpline(is_temp, is_vom)

xs = np.linspace(1.5, 4.0, 1000)

mag_spl.set_smoothing_factor(3)
plt.plot(is_temp,is_mag,'r-',)
plt.plot(xs, mag_spl(xs), 'g-',lw=3)
plt.xlabel('Temperature',fontsize=16)
plt.ylabel('Magnetization',fontsize=16)
plt.grid()
plt.show()

stdev_spl.set_smoothing_factor(0.3)
plt.plot(is_temp,is_stdev,'b-')
plt.plot(xs, stdev_spl(xs), 'g-',lw=3)
plt.xlabel('Temperature',fontsize=16)
plt.ylabel('Mag Stdev',fontsize=16)
plt.grid()
plt.show()

vom_spl.set_smoothing_factor(0.1)
plt.plot(is_temp,is_vom,'y-')
plt.plot(xs, vom_spl(xs), 'g-',lw=3)
plt.xlabel('Temperature',fontsize=16)
plt.ylabel('VoM',fontsize=16)
plt.grid()
plt.show()

```

A.3.5 The final algorithm

```
# Parameters of our fake system
J = 1
B = 0
N = 64
d = 2

repeats = 1000

Temps = np.linspace(1.5,3.5,160)
fake_list = []
F1s, sebs, var_difs, sk_difs, kr_difs, voms = [], [], [], [], [], []
Es, Ms, Cs, Xis= [], [], [], []
Mstds = []
pathx, pathy = [], []

for T in Temps:
  F1_list, seb_list, var_dif_list, sk_dif_list, kr_dif_list, vom_list = [], [], [], [], [], []
  Ei, Mi = [], []
  pathx_list, pathy_list = [],[]

  for i in range(repeats):
  mag_spl.set_smoothing_factor(3)
  mag = mag_spl(T)
  stdev_spl.set_smoothing_factor(0.3)
  st_dev = stdev_spl(T)
  vom_spl.set_smoothing_factor(0.1)
  vom = vom_spl(T)
  m = normal(loc=-np.abs(mag), scale=d*st_dev, size=1)
  # m = [-np.abs(mag)]
  X = np.array([[T,m[0],vom]])
  if T<2.1 or T>2.4:
    p = MLP.predict(X)
  else:
    p = critical.predict(np.array([[m[0],vom]]))
  p=p[0]
  pathx_list.append(p[0])
  pathy_list.append(p[1])

  img = SAimage_fft_2(N_total=64,rms=3,skewness=1,kurtosis=3,corlength_x=p[0],corlength_y=p[0],alpha=0.6,th=p[1],non_Gauss
 =False,off=True)

  lattice = binary_to_ising(img)
  coord, label = image_to_dataset(lattice)
  idx, F1 = dcm.F1(coord, label)

  F1_list.append(F1)

  Ei.append(getE(lattice ,N, J, B))
  Mi.append(getM(lattice))
  seb_list.append(seb_index(lattice))
  vom_list.append(var_of_mean(lattice))
  var_dif_list.append(var_diff(lattice))
  sk_dif_list.append(skew_diff(lattice))
  kr_dif_list.append(kurt_diff(lattice))

# create path vectors
pathx.append(np.mean(pathx_list))
pathy.append(np.mean(pathy_list))

# C and Xi computed from energies
Cs.append(getC(Ei, T))
```

```
61      Xis.append(getXi(Mi, T))
62
63      # Compute the means of all the other stuff
64      F1s.append(np.mean(F1))
65      Es.append(np.mean(Ei))
66      Ms.append(np.mean(Mi))
67      Mstds.append(np.var(Mi))
68      sebs.append(np.mean(seb_list))
69      voms.append(np.mean(vom_list))
70      var_difs.append(np.mean(var_dif_list))
71      sk_difs.append(np.mean(sk_dif_list))
72      kr_difs.append(np.mean(kr_dif_list))
73
74      fake_list.append(img)
75
```

Bibliography

[1] M. E. J. Newman, G. T. Barkema, *Monte Carlo Methods in Statistical Physics*, Oxford University Press 2001.

[2] S. Friedli, Y. Velenik, *Statistical Mechanics of Lattice Systems: A Concrete Mathematical Introduction*, `http://www.unige.ch/math/folks/velenik/smbook/`.

[3] Barry M. McCoy, Tai Tsun Wu,*The Two-Dimensional Ising Model*, Dover Publications, Inc. Mineola, New York 2014.

[4] Yang Tang, Jürgen Kurths, Wei Lin, Edward Ott, and Ljupco Kocarev, *Introduction to Focus Issue: When machine learning meets complex systems: Networks, chaos, and nonlinear dynamics*, Chaos, 5 June 2020.

[5] Peter Eastman, *Introduction to Statistical Mechanics*, Stanford University 2014-2015, `https://web.stanford.edu/~peastman/statmech/index.html`.

[6] Chjan Lim, Joseph Nebus, *Vorticity, Statistical Mechanics, and Monte Carlo Simulation*, Springer Monographs in Mathematics, 2007 USA New York

[7] Richard Leach, *Characterisation of Areal Surface Texture*, Springer-Verlag Berlin Heidelberg 2013.

[8] Christopher M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006 USA.

[9] Richard O. Duda, Peter E. Hart, David G. Stork, *Pattern Classification*, Wiley-Interscience, November 9, 2000.

[10] Ana C. Lorena, Luis P. F. Garcia, Jens Lehnmann, Marcilio C. P. Souto, Tim Kam Ho, *How Complex Is Your Classification Problem?: A Survey on Measuring Classification Complexity*, ACM Computing Surveys, Vol. 52, No. 5, Article 107. Publication date: September 2019.

[11] R. Wackerbauer, A. Witt, H. Atmanspacher, J. Kurths, H. Scheingraber, *A Comparative Classification of Complexity Measures*, Chaos, Solitons & Fractals, Vol. 4, No. 1, pp. 133-173, 1994.

[12] A.N. Kolmogorov, *Three approaches to the quantitative definition of information*, Inf. Trans. 1, 3-11 (1965).

[13] G. Chaitin, *Algorithmic information theory*, Cambridge University Press, Cambridge (1987).

[14] Andrey A. Bagrov, Askar A. Iliasov, Ilia A. Iakovlev, Mikhail I. Katsnelson, and Vladimir V. Mazurenko, *Multiscale structural complexity of natural patterns*, December 1, 2020, www.pnas.org/cgi/doi/10.1073/pnas.2004976117.

[15] Guoqing Yang, Baotong Li, Yang Wang and Jun Hong, *Numerical Simulation of 3D Rough Surfaces and Analysis of Interfacial Contact Characteristics*, 2014 Tech Science Press, CMES, vol.103, no.4, pp.251-279, 2014.

[16] Albert-Laszlo Barabasi, H. Eugene Stanley, *Fractal Concepts in Surface Growth*, Cambridge University Press, January 1995.

[17] Yootai Kim, Raghu Machiraju, David Thompson, *Modeling Rough Surfaces*, 2015.

[18] James Ladyman and Karoline Wiesner: What Is a Complex System?, *Essay Review: Complexity Features, (Putative) Truisms, and the Ising Model*, 11 January 2021, Journal for General Philosophy of Science https://doi.org/10.1007/s10838-021-09554-6.

[19] Antonios Haralabos Stellas, Dr. V. Constantoudis, Dr. I. Giannakopoulos, *Machine Learning and Nanotechnology: A Connection Between Structural and Functional Parameters of Nanostructured Rough Surfaces*, National and Technical University of Athens, Master of Nanotechnology and Nanosystems, Athens 2018.

[20] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, Lenka Zdeborová, *Machine learning and the physical sciences*, Reviews of Modern Physics, Volume 91, October-December 2019.

[21] Iasonas Kokkinos, *Introduction to Machine Learning*, course slides, University College London.

[22] John Hertz, Anders Krogh, Richard G. Palmer, *Introduction to the Theory of Neural Computation*, 2018, CRC Press, Taylor and Francis Group.

[23] Tassos Bountis, *Lectures in Complex Hamiltonian Dynamics*, January 27, 2013, Springer.

[24] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, David J. Schwab, *A high-bias, low-variance introduction to Machine Learning for physicists*, Science Direct, Elsevier, 2 February 2019.

[25] Apostolos Karampakos, *The Relation of Renormalization Group and Machine Learning in Spin Models*, Master Thesis, 14 September 2020, National and Kapodistrean University of Athens.

[26] Pascal Monceau and Michel Perreau, *Critical behavior of the Ising model on fractal structures in dimensions between one and two: Finite-size scaling effects*, Physical Review B, Volume 63, 19 April 2001.

[27] J.R. Norris, *Markov Chains*, Cambridge University Press 1997.

[28] Michalis Loulakis, *Stochastic Processes*, notes from his master course, https://repository.kallipos.gr/handle/11419/6003.

[29] Lewrence Perko, *Differential Equations and Dynamical Systems*, Springer 2001.

[30] Steven H. Strogatz, *Nonlinear Dynamics and Chaos: With Applications to Physics, Biology, Chemistry and Engineering*, Perseus Books.

[31] Artem Oppermann, *Deep Learning meets Physics: Restricted Boltzmann Machines Part I (II)*, 27 Apr 2018, `https://towardsdatascience.com/deep-learning-meets-physics-restricted-boltzmann-machines-part-i-6df5c4918c15`.

[32] Shotaro Shiba Funai and Dimitrios Giataganas, *Thermodynamics and Feature Extraction by Machine Learning*, 2018.

[33] Ellen De Mello Koch, Robert De Mello Koch, and Ling Cheng, *Is Deep Learning a Renormalization Group Flow?*, 2020.

[34] Alan Morningstar, *Deep Learning the Ising Model Near Criticality*, Roger G. Melko, Journal of Machine Learning Research, Published 4/18.

[35] Cédric Bény, *Deep learning and the renormalization group*, arXiv, 13 Mar 2013.

[36] Ken-Ichi Aoki, and Tamao Kobayashi, *Restricted Boltzmann machines for the long range Ising models*, 1 December 2016, Modern Physics Letters B, World Scientific.

[37] Chris Williams, Felix Agakov, *An Analysis of Contrastive Divergence Learning in Gaussian Boltzmann Machines*, Division of Informatics, Institute for Adaptive and Neural Computation.

[38] Davit Potoyan, *Ising models and Metropolis-Hastings algorithm*, 2021, `https://dpotoyan.github.io/Statmech4ChemBio/06_ising/01_MCMC_Ising.html`.

[39] Aaron Griffith, Andrew Pomerance, and Daniel J. Gauthier, *Forecasting Chaotic Systems with Very Low Connectivity Reservoir Computers*, 19 November 2019, arXiv.

[40] Qunxi Zhu, Huanfei Ma, and Wei Lin, *Detecting unstable periodic orbits based only on time series: When adaptive delayed feedback control meets reservoir computing*, 24 September 2019.

[41] Thomas Lymburn, David M. Walker, Michael Small, and Thomas Jüngling, *The reservoirs perspective on generalized synchronization*, Chaos, 30 September 2019.

[42] A. Cunillera, M. C. Soriano, and I. Fischer, *Cross-predicting the dynamics of an optically injected single-mode semiconductor laser using reservoir computing*, Chaos, 13 November 2019.

[43] Guorui Shen, Jürgen Kurths, and Ye Yuan, *Sequence-to-sequence prediction of spatiotemporal systems*, Chaos, 13 January 2020.

[44] Martin Lellep, Jonathan Prexl, Moritz Linkmann, and Bruno Eckhardt, *Using Machine Learning to predict extreme events in the Hénon map*, February 25 2020, arXiv.

[45] Pablo Amil, Miguel C. Soriano, and Cristina Masoller, *Machine learning algorithms for predicting the amplitude of chaotic laser pulses*, Chaos, 12 November 2019.

[46] Rosangela Follmann and Epaminondas Rosa, *Predicting slow and fast neuronal dynamics with machine learning*, Jr., Chaos, 18 November 2019.

[47] Ren-Meng Cao, Si-Yuan Liu, and Xiao-Ke Xu, *Network embedding for link prediction: The pitfall and improvement*, Chaos, 08 October 2019.

[48] Paul A. Gagniuc, Constantin Ionescu-Tirgoviste, Elvira Gagniuc, Manuella Militaru, Lawrence Chukwudi Nwabudike, Bujorel Ionel Pavaloiu, Andrei Vasileanu, Nicolae Goga, George Drgoi, Irinel Popescu, and Simona Dima, *Spectral forecast: A general purpose prediction model as an alternative to classical neural networks*, Chaos 10 March 2020.

[49] Sandip V. George, R. Misra, and G. Ambika, *Classification of close binary stars using recurrence networks*, Chaos, 13 November 2019.

[50] Qi Ni, Jie Kang, Ming Tang, Ying Liu, and Yong Zou, *Learning epidemic threshold in complex networks by Convolutional Neural Network*, Chaos, 06 November 2019.

[51] Sebastian J. Wetzel, *Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders*, Physical Review, 18 August 2017.

[52] Alex J. Smola and Bernhard Schölkopf, *A Tutorial on Support Vector Regression*, September 30, 2003.

[53] Kamalika Chaudhuri, *Nearest Neighbors I: Regression and Classification*, slides, University of California, San Diego.

[54] Andreas Athenodorou, *Unsupervised Recognition of Phase Transition on the Lattice*, Slides, 8th International Conference on New Frontiers in Physics (ICNFP 2019), Kolybari, 27th of August 2019.

[55] Georgios Drakos, *Decision Tree Regressor explained in depth*, GDCoder, 23 May 2019, `https://gdcoder.com/decision-tree-regressor-explained-in-depth/`.

[56] Multilayer Perceptron, Dive into Deep Learning, `https://classic.d2l.ai/chapter_multilayer-perceptrons/mlp.html`.

[57] Basics of Multilayer Perceptron A Simple Explanation of Multilayer Perceptron, Genius Blog, January 7, 2018, `https://kindsonthegenius.com/blog/basics-of-multilayer-perceptron-a-simple-explanation-of-multilayer-perceptron/`.

[58] The Ultimate Guide to Random Forest Regression, September 17 2020, `https://www.keboola.com/blog/random-forest-regression`.

[59] Wikipedia, *Linear regression*, `https://en.wikipedia.org/wiki/Linear_regression`.

[60] Wikipedia, *Reinforcement learning*, `https://en.wikipedia.org/wiki/Reinforcement_learning`.