Alma Mater Studiorum · Università di Bologna

Scuola di Scienze
Dipartimento di Fisica e Astronomia
Corso di Laurea Magistrale in Fisica

Deep Learning
Topological Phases of Matter

Relatore:

Prof.ssa Elisa Ercolessi

Correlatore:

Dott. Davide Vodola

Presentata da:

Simone Tibaldi

Anno Accademico 2018/2019

Abstract

Lo scopo di questo lavoro di tesi è di presentare l'implementazione di un problema di fisica della materia condensata con un approccio di Deep Learning. Per farlo introdurremo tre modelli che presentano caratteristiche topologiche: il modello di Kitaev (un superconduttore unidimensionale) con accoppiamento tra primi vicini, secondi vicini e interagente. In seguito presenteremo le tecniche di Machine Learning necessarie per svolgere il compito che ci siamo posti. Infine applicheremo tali tecniche per allenare una rete neurale deep e una convoluzionale a riconoscere le fasi topologiche del modello non interagente e predire le fasi di quello interagente.

Abstract

This thesis is aimed at showing how to set up a typical problem of Condensed Matter physics in a Deep Learning framework. In order to do this we will introduce the Kitaev model (a superconducting quantum wire with topological properties) with nearest neighbor coupling, next to nearest neighbor coupling and an interacting term. Then we will present the Machine Learning techniques we are going to use. Finally we will apply them to train a Neural Network and a Convolutional Neural Network on recognizing the topological phases of matter of the non-interacting model to test it on the classification of interacting data.

Contents

Introduction

Condensed Matter physics relies on collective phenomena, that is the occurrence of a global behaviour of many interacting bodies. In particular, phase transitions are typical manifestations of a collective phenomenon. The study of phase transitions has had at its core the Ginzburg-Landau theory for many decades, based on Spontaneous Symmetry Breaking. This picture changed with the discovery of the Quantum Hall Effect [1] which led to the discovery of a new type of phase transition where the symmetry is not changed. That is, the topological phase transition [2].
According to the topological theory of phases of matter, there are several global characteristics of a system for which two states can be in the different topological phases while being in the same symmetry phase. Thus, the topological classification of phases of matter goes beyond the symmetry classification [3]. An example of model with an interesting topological behaviour is the Kitaev chain [4], a one dimensional superconducting chain.
Machine Learning is an ever growing, extremely popular field of study which also exploits the concept of a collective phenomen. Its purpose is to create algorithms capable of learning from experience (data) and in order to do so it needs to rely on structures with so many parameters to capture the complexity of any system [5]. That is brought to an even more abstract level by Deep Learning, a subset of Machine Learning, where the architectures of the algorithms are so complex that they can capture the features describing the data and learn from them [6].
In the last few years Machine Learning and Deep Learning have flourished thanks to the enormous amount of data available to learn from and their success in many tasks from image and speech recognition to self-driving cars [5]. This has led to their application to every field of science, including physics [7]. This is in fact justifiable because Machine Learning relies on many mathematical concepts borrowed from physics, starting from the first real premise of being able to learn from experience.
In the last few years, Condense Matter phyisicists have paid a lot of attention to

study the potential applicability of Machine Learning [8]. It turned out to be useful for a wide variety of tasks from encoding the wavefunction and the dynamics of a many body system [9], [10], to the study of classical and quantum phase transitions which is the main goal of this theis. For example, Deep Learning was used to study the phase transitions of classical 2D Ising Model [11], [12] and in lattice gauge theory [12], for the Kitaev and Heisenberg models [13], in general topological models with a specific symmetry [14], [15], and many other models with topological behaviour [8].
As it is the case for fast growing fields of study, there has not been enough theoretical background supporting the development of Machine Learning applied to science. It is therefore now part of the effort of the physics community to analyze more in-depth how this algorithms work and why they are capable of making sensible decisions [7]. Our work can be placed into this context and our aim is twofold: on one hand we want to apply Machine and Deep Learning techniques to get insight into our data and classify the topological phases of matter it belongs to. On the other hand, we are interested in showing the behaviour of the algorithms. We want to do so by training two different neural networks: a fully connected feed forward and a convolutional. We train and test them on data obtained from the non-interacting Kitaev chain. Then test them on data generated with the interacting Kitaev chain to see that the convolutional neural network has more generalization power then the fully connected one.
We care to stress that the purpose of this work is not to study the efficiency of Machine Learning applied to physics but more to gain insight on how to set up a modern and interesting physics problem in the Machine Learning framework. For these reasons this thesis was structured in this way:

- Chapter 1. Here we introduce the physical framework we are talking about. We will present the topological classification of phases of matter and why it is a relevant field of study. After that we will introduce three models which show a topological behaviour. Firstly, the Kitaev model, a one dimensional superconducting fermionic chain. Then, two variations of this model, one with next to nearest neighbor coupling and one with an interacting term in the Hamiltonian.

- Chapter 2. The second chapter is dedicated to present all the concepts and methods necessary to work with Machine Learning. We will do so trying to emphasize aspects that can be interesting under a physical point of view while discarding more technical aspects, for which we refer to other sources. Therefore, we will start by introducing the basic concepts needed for Machine Learning. Then, we will move to the presentation of the Deep Learning models we implemented for our study. Lastly, we will present two examples of applying Machine Learning techniques to study the classification of phases of matter on the Ising 2D Classical Model and the Kitaev 1D Topological Model.

- Chapter 3. The last chapter is dedicated to the new results. The main idea is to train a Neural Network to classify topological phases of the non-interacting Kitaev model and test it on data generated with the interacting model. This includes applying a preprocessing technique (Principal Components Analysis) to learn the main features of our data. Then, we apply a Neural Network and a Convolutional Neural Network to solve the problem. As already mentioned, our purpose is just to present a typical Condensed Matter physics problem approached with Machine Learning, for this reason we will concentrate on showing what is learnt inside the algorithms and less on the training procedure.

1 Topological Phases of Matter

Classification of phases of matter has always been one of the main goals of Condensed Matter Physics. The Ginzburg-Landau theory has been the corner stone of this research field and it is based on the mechanism of *Spontaneous Symmetry Breaking*, in a sense that two phases differ in their symmetric, local characteristics such as the order parameters [16]. The discovery of the Quantum Hall Effect (1980) [1] and its study in the following decades has led to a new type of classification based on the notion of *topological order*. That is because the QHE manifests in a state without the need of breaking any symmetry, but rather because the state has fundamental characteristics that are insensible to smooth changes in the system's parameters. The only way to change this characteristic is for the system to undergo a quantum phase transition.

The study of the QHE led to the development of the research field of Topological Phases of Matter. This field is based on a new classification of phases that takes into account symmetry but goes further because it studies systems where there can be a phase transition even without breaking any symmetry. The word *topological* is indeed borrowed from topology, the field of mathematics that deals with the broader, non-local, characteristics of a system that do not change through continuous deformations of the system.

In this introductory chapter we first present the fundamental concepts of topological matter, that is the symmetry classification of free fermionic Hamiltonians and the concept of topological invariants.

We then move to the main model of this thesis, the Kitaev chain [4] which will be studied in its original form, a variation with a next-to-nearest neighbor coupling and its interacting form.

1.1 Symmetry Classification of Topological Phases

The concept and the field of topological phase transitions, as mentioned above, were born due to the study of particular systems in which symmetries played no role and yet a phase transition could occur. Anyhow, symmetries remain a fundamental concept even in the context of topological phases for they still are protagonists of many aspects that characterize a topological phase and transition.

In this section we want to show how symmetry can be the tool to classify topological phases of matter.

The Hamiltonians we are interested in, and that we will be dealing with throughout this whole chapter, are at temperature $T = 0$. Being the system at zero temperature, the ground state is assumed to be the equilibrium state. We also assume these Hamiltonians to depend on some parameter g (*i.e.* $H(g)$). Let us consider the expectation value of some operator O on the ground state of these Hamiltonians. We can thus say that if, in the infinte limit system size, there is a discontinuity for some parameter g_c of $\langle O \rangle$ then the system underwent a quantum phase transition [16].

By defining a phase transition we can infer when two states are in the same phase. That is, given two groundstates $|\phi(0)\rangle$, $|\phi(1)\rangle$ of different Hamiltonians $H(0)$ and $H(1)$, if we can find a continuous path $H(g)$ $(0 \leq g \leq 1)$ connecting $H(0)$ to $H(1)$ without a phase transition, the two groundstates are in the same phase.

Lastly, these Hamiltonians are *gapped.* That means there is a finite energy difference between the ground state and the first excited state of the system. The presence of a gap makes it non-trivial to find the Hamiltonians that are connected through a continuous transformation. In fact we can say that the two states $|\phi(0)\rangle$, $|\phi(1)\rangle$ are in the same phase if the gap of $H(g)$ does not close on the continuous path from $H(0)$ to $H(1)$. When this happens we say that the two groundstates are in the same *topological phase.*

We need to look for particular properties of the ground states separated by a quantum phase transition. If this properties are topological, meaning that they do not depend on local characteristics or symmetry arguments, they can define ground states in different topological phases.

Another equivalence relation for states in the same phase can be obtained. We can say that two systems are in the same phase if they are connected through a Local

Unitary (LU) evolution [16], that is

$$|\phi(0)\rangle \sim |\phi(1)\rangle \qquad \text{iff} \qquad |\phi(1)\rangle = \mathcal{T}[e^{-i\int_0^1 dg\tilde{H}(g)}] \tag{1.1}$$

where $\mathcal{T}$ is the time-ordering operator and $\tilde{H}(g)$ is a sum of local Hermitean operators.

Introducing the LU is useful to define a first classification of phases of matter based on entanglement. In fact a state that can be transformed into a direct-product or *unentangled* state by a LU transformation such as (1.1) is said to have Short Range Entanglement (SRE). The opposite case, defines a state with Long Range Entanglement (LRE). LRE states turn out to have what is called *intrinsic bulk topological order* [3]. That is, they possess a series of properties such as anyonic excitations or groundstate degeneracies. On the other hand, SRE can all be connected to each other trough a LU (since they can all be connected to a direc-product state) and are thus all in the same phase.

Although there seems to be two different classes of topological phases of matter - namely, the ones who possess intrinsic topological order and the ones characterized by SRE - we have not considered symmetries yet.

Adding symmetries adds richness to the classification of phases. We can identify three cases [3]: SRE states arising by spontaneous symmetry breaking, SRE states where the symmetry is not broken called Symmetry Protected Topological (SPT) states and LRE states with given symmetry constraints, called Symmetry Enhanced Topological (SET) phases. Since our interest are the difference between SRE to LRE and the presence of symmetry we summerize the main properties of Intrinsic Topological (LRE) states and Symmetry Protected Topological (SRE) states.

Let us see them:

- **Intrinsic Topological Order.** As already mentioned long-range entanglement is the main feature of this class of topological phases. The presence of a symmetry does not play any special role and it is thus not relevant for their classification [3].

  These states show topological order bulk properties such as ground state degeneracies on topologically non-trivial manifolds, anyonic excitations which may have fractional quantum numbers and robust gapless edge excitations.

- **Symmetry Protected Topological States.** They are characterized by SRE and by a symmetry group for which all states are invariant. That is, there is no

symmetry breaking in the bulk to classify these states. Even though they have a bulk gap they have no interesting bulk properties, but show their topological nature on their boundaries.
The boundaries must have one of this properties [3]:

- Spontaneously break the symmetry governing the phase.
- In the presence of a gap, must have intrinsic (boundary) topological order, meaning it has to present features such as anyonic excitations.
- Be gapless, which is understood as the presence of a zero energy state at the boundary.

Examples of systems with intrinsic topological order are the fractional quantum Hall or the toric code, both LRE states with familiar properties of a topological order. Symmetry Protected state are the Kitaev p-wave superconductor or the Integer quantum Hall. In the former, as an example, we will see that it shows zero energy excitations at the edges of the one-dimensional case, manifesting its boundary topological properties.
It is important once again to state what the word *topological* means in the way it was used in this section. As much as in the same field of mathematics, topological here means related to the global properties of the system that are preserved under smooth transformations of its parameters. That is, we do not care about local differences in Hamiltonians but only on general characteristic that are in common between the different phases.

1.1.1 Role of Symmetries

We now turn to the symmetry classification of non-interacting fermionic Hamiltonians. By Symmetry classification we mean that we look at the least trivial and most defining symmetries that a system can possess and how they can be used to split the systems into different topological categories.
The study is only focused on free Hamiltonians because we have a full classification of these systems, known as the *ten-fold way* [3]. Moreover, we have analytical solutions of these systems and they can provide a starting point to study interacting models.

Let us start by writing a general non-interacting fermionic Hamiltonian in the form

$$\mathcal{H} = \sum_{a,b} \Psi_a^\dagger H_{ab} \Psi_b \tag{1.2}$$

where Ψ_α are fermionic operators acting on a Fock space as much as $\mathcal{H}.[H_{ab}] = H$ is a Hermitean matrix that gives the single particle representation of the Hamiltonian. A symmetry is realized by its operators that act on a Hilbert space and leave invariant the modulus of the scalar product of two Hilbert states. By Wigner theorem, those operators can be either unitary or antiunitary. In this part we mainly follow the work of [3].

Unitary and Antiunitary Operators A surjective transformation $\mathcal{U}$ defined on a Hilbert space is unitary if

$$\langle \mathcal{U}\Psi | \mathcal{U}\Phi \rangle = \langle \Psi | \Phi \rangle \tag{1.3}$$

and it is a symmetry of a system if it commutes with the Hamiltonian $\mathcal{H}$

$$\mathcal{U}\mathcal{H}\mathcal{U}^\dagger = \mathcal{H} \tag{1.4}$$

On a Fock space it acts on the fermionic operators as a change of basis:

$$\mathcal{U}\Psi_a\mathcal{U}^\dagger = \sum_c U_{ac}^\dagger \Psi_b \qquad\qquad \mathcal{U}\Psi_a^\dagger\mathcal{U}^\dagger = \sum_d \Psi_d^\dagger U_{db}^\dagger \tag{1.5}$$

We can calculate its action on $\mathcal{H}$ in the following way

$$\mathcal{U}\mathcal{H}\mathcal{U}^\dagger = \mathcal{U}\sum_{a,b} \Psi_a \mathcal{U}^\dagger \mathcal{U} H_{ab} \mathcal{U}^\dagger \mathcal{U} \Psi_b \mathcal{U}^\dagger = \sum_{c,d,a,b} \Psi_c U_{ca} H_{ab} U_{bd}^\dagger \Psi_d = \sum_{c,d} \Psi_c [U \cdot H \cdot U^\dagger]_{cd} \Psi_d^\dagger \tag{1.6}$$

which clearly shows that if the single particle Hamiltonian transforms as

$$H = U \cdot H \cdot U^\dagger \tag{1.7}$$

then the Hamiltonian $\mathcal{H}$ is invariant under the symmetry $\mathcal{U}$. We know that this means that the Hamiltonian can be put in a block-diagonal form, with each block associated to a quantum number which is not a universal property of the system. That is why unitary transformations commuting with H in the single particle frame are not interesting to us even though the principal symmetries, (translations, rota-

tions etc...) are in fact represented by unitary transformations [3]. Therefore, we now turn our attention to antiunitary symmetries.
An antiunitary transformation, instead, acts on two Hilbert states like this

$$\langle \mathcal{A}\Psi|\mathcal{A}\Phi\rangle = \langle \Psi|\Phi\rangle^* = \langle \Phi|\Psi\rangle \tag{1.8}$$

while on complex numbers it is antilinear, meaning

$$\mathcal{A}\alpha\mathcal{A}^{-1} = \alpha^* \tag{1.9}$$

The symmetries produced by these operators are particular interesting for the classification of free fermionic Hamiltonians. Let us see them in detail.

Time Reversal Symmetry We consider time reversal $\mathcal{T}$ as a change of the time parameter $t \to -t$ so that it flips the momentum operator sign, leaving the position operator unchanged. It acts on the fermionic operators in the same way as a unitary transformation

$$\mathcal{T}\Psi_a\mathcal{T} = \sum_c U^\dagger_{ac}\Psi_c \qquad\qquad \mathcal{T}\Psi_b\mathcal{T} = \sum_d \Psi^\dagger_d U_{db} \tag{1.10}$$

to preserve the anti-commutation relations. On a Fock space Hamiltonian it acts as

$$\begin{aligned}\mathcal{T}\mathcal{H}\mathcal{T}^{-1} &= \mathcal{T}\sum_{a,b}\Psi_a\mathcal{T}^{-1}\mathcal{T}H_{ab}\mathcal{T}^{-1}\mathcal{T}\Psi_b\mathcal{T}^{-1} = \sum_{c,d,a,b}\Psi_c U_{ca}\mathcal{T}H_{ab}\mathcal{T}^{-1}U^\dagger_{bd}\Psi_d \\ &= \sum_{c,d}\Psi_c[U\cdot H^*\cdot U^\dagger]_{cd}\Psi^\dagger_d\end{aligned} \tag{1.11}$$

Notice that since $\mathcal{T}$ is antiunitary, its action on the time evolution operator $U(t) = \exp\{itH\}$ is $\mathcal{T}U(t)\mathcal{T}^{-1} = U(-t)$ as we expect from a time-reversal operator.
From (1.11) we see that a system is invariant under time reversal transformation if

$$THT^{-1} = H \tag{1.12}$$

where T is the first quantized version of $\mathcal{T}$ and we define $T = U_TK$ with K complex conjugation operator such that $KHK^{-1} = H^*$ and U_T a unitary matrix so that T is antiunitary.
The square of the time reversal operator $\mathcal{T}^2$ is unitary and its matrix representation is $T^2 = U_TU_T^*$. Acting twice with the time reversal operator means going back

to the original configuration, thus we expect its square to be one or at most to be proportional to a phase. It can be shown [3] that this leaves us three possible situations that we will label with $\mathcal{T}$:

$$\mathcal{T} = \begin{cases} \varnothing, & \text{The system is not time-reversal invariant} \\ +1, & \text{The system is time-reversal invariant and } T^2 = +1 \\ -1, & \text{The system is time-reversal invariant and } T^2 = -1 \end{cases} \tag{1.13}$$

Particle-Hole Symmetry It is another word for charge conjugation symmetry, which is the symmetry that turns fermions into antifermions and viceversa. On a Fock space it acts in the following way

$$\mathcal{C}\Psi_a\mathcal{C}^{-1} = \sum_c (U^\dagger_{ac})^*\Psi^\dagger_a \qquad\qquad \mathcal{C}\Psi^\dagger_b\mathcal{C}^{-1} = \sum_d \Psi_d(U_{db})^* \tag{1.14}$$

and it is a unitary Fock space operator, *i.e.* $\mathcal{C}i\mathcal{C}^{-1} = i$.
The Fock space Hamiltonian $\mathcal{H}$ is invariant under charge conjugation, that is

$$\mathcal{C}\mathcal{H}\mathcal{C}^{-1} = \mathcal{H} \tag{1.15}$$

if its single-particle representation transforms as

$$CHC^{-1} = -H \tag{1.16}$$

where $C = U_C K$, being K charge conjugation operator and U_C unitary. Repeating the same argument of the time reversal operator, acting twice with charge conjugation brings the system in its original form thus we can expect three configurations for C and thus the label of the symmetry $\mathcal{C}$:

$$\mathcal{C} = \begin{cases} \varnothing, & \text{The system is not particle-hole invariant} \\ +1, & \text{The system is particle-hole invariant and } C^2 = +1 \\ -1, & \text{The system is particle-hole invariant and } C^2 = -1 \end{cases} \tag{1.17}$$

Chiral Symmetry There is yet another type of symmetry that can be constructed. Since a general symmetry in the single particle space is unitary and commuting, time reversal is antiunitary and commuting, particle hole is antiunitary and anticommuting we can expect a unitary anticommuting (in the single particle representation) symmetry in principle.

Let us suppose such symmetry exists, if it anti-commutes with the single particle Hamiltonian it is such that

$$SHS^{\dagger} = -H \tag{1.18}$$

In order for this relation to work we need to impose the following transformation rules for the fermionic operators

$$\mathcal{S}\Psi_a\mathcal{S}^{-1} = \sum_c (U^*)^{\dagger}_{ac}\Psi^{\dagger}_c \qquad \mathcal{S}\Psi_b\mathcal{S}^{-1} = \sum_d \Psi_d U^*_{db} \tag{1.19}$$

along with $\mathcal{S}i\mathcal{S}^{-1} = -i$ making it antiunitary on the Fock space and commuting with the second quantized Hamiltonian $\mathcal{S}\mathcal{H}\mathcal{S}^{-1} = \mathcal{H}$.
Chiral symmetry is not a symmetry *per se* because it anticommutes with $\mathcal{H}$. It is still considered a symmetry traditionally and it is also called *sublattice symmetry* because it arises naturally in systems that are divided into two parts which interact with each other and not with themselves.
Interestingly, chiral symmetry is not independent of charge conjugation and time reversal. It is indeed made out of the product of the two in the single particle space whenever H is invariant for both T and C. That is, $S = TC = U_T U_C^*$. It can be shown [3] that S^2 and thus its label $\mathcal{S}$ can only take two values

$$\mathcal{S} = \begin{cases} \varnothing, & \text{When } H \text{ does not anticommute with chiral symmetry S} \\ +1, & \text{When H anti commutes with chiral symmetry S} \end{cases} \tag{1.20}$$

1.1.2 Ten-fold Classification

The classification of free fermionic Hamiltonians can be done considering combinations of only time-reversal, charge conjugation and chiral symmetry. It is important to stress that these three symmetries are not randomly picked but they are the only possible symmetries realizable on a Hamiltonian after we eliminated all symmetries generated by a unitary operator.
That is, we can consider 2 cases with only $\mathcal{T}$, and 2 with only $\mathcal{C}$ where $\mathcal{S}$, then, cannot be realized. Then we have 4 cases with both $\mathcal{T}$ and $\mathcal{C}$ and thus a single fixed value of $\mathcal{S}$ obtained from their product, one case with only $\mathcal{S}$ and finally the last case without any symmetry. This adds up to 10 possible combination, listed in table 1.1. The symmetry classes are named after a classification given by E. Cartan. The table also shows what class the evolution operator $U(t)$ is according to the symmetries respected by H. We want to show now how having different symmetries for a

Cartan label	$\mathcal{T}$	$\mathcal{C}$	$\mathcal{S}$	$\exp itH$
A (unitary)	$\varnothing$	$\varnothing$	$\varnothing$	U(N)
AI (orthogonal)	$+1$	$\varnothing$	$\varnothing$	$\mathrm{U}(N)/\mathrm{O}(N)$
AII (sympletic)	-1	$\varnothing$	$\varnothing$	$\mathrm{U}(2N)/\mathrm{Sp}(2N)$
AIII (ch. unit.)	$\varnothing$	$\varnothing$	1	$\mathrm{U}(N+M)/\mathrm{U}(N)\times\mathrm{U}(M)$
BDI (ch. orth.)	$+1$	$+1$	1	$\mathrm{O}(N+M)/\mathrm{O}(N)\times\mathrm{O}(M)$
CII (ch. sympl.)	-1	-1	1	$\mathrm{Sp}(N+M)/\mathrm{Sp}(N)\times\mathrm{Sp}(M)$
D (BdG)	$\varnothing$	$+1$	$\varnothing$	$\mathrm{SO}(2N)$
C (BdG)	$\varnothing$	-1	$\varnothing$	$\mathrm{Sp}(2N)$
DIII (BdG)	-1	$+1$	1	$\mathrm{SO}(2N)/\mathrm{U}(N)$
CI (BdG)	$+1$	-1	1	$\mathrm{Sp}(2N)/\mathrm{U}(N)$

Table 1.1: Topology classes and their time operator evolution space. The Cartan labels and their associated symmetries are shown. With that, the space of the time evolution operator e^{-itH} is provided. For example, class A does not have any symmetry so the time evolution is provided by a single particle Hamiltonian H that only needs to be Hermitean. Thus, e^{-itH} is unitary and so the time operator evolution space is U(N).

system leads to particular behviours of the ground states of said system. In order to do that we consider translational invariant systems for simplicity. This lets us write the Schrödinger equation in momentum space

$$\boldsymbol{H}(\boldsymbol{k})|u_\alpha(\boldsymbol{k})\rangle = E_\alpha(\boldsymbol{k})|u_\alpha(\boldsymbol{k})\rangle \tag{1.21}$$

where α runs on all the bands and $\boldsymbol{k}$ is a D dimensional vector on a D-dimensional Brillouin Zone.

Our purpose here is to write all the translational invariant Hamiltonians into a simplified form where all the n occupied bands have an assigned energy of -1 and the m empty ones have an energy of $+1$.

We can do so by using a projector on the filled bands $P(\boldsymbol{k})$ so that

$$Q(\boldsymbol{k}) = \mathbb{1} - 2P(\boldsymbol{k}) \tag{1.22}$$

is the simplified, also named *flatband*, Hamiltonian we were looking for and it satisfies

$$Q(\boldsymbol{k})^\dagger = Q(\boldsymbol{k}) \qquad Q(\boldsymbol{k})^2 = \mathbb{1} \qquad \mathrm{Tr}[Q(\boldsymbol{k})] = m - n \tag{1.23}$$

The new Hamiltonian $Q(\boldsymbol{k})$ still holds the topological information of the original H and it can be seen as a mapping from the BZ to a space $\mathcal{S}$. Both the the BZ and $\mathcal{S}$ are topological spaces, specifically due to the map connecting them the BZ is called the *base space* and $\mathcal{S}$ the *target space*.
At this point it is important to introduce the concept of *homotopy*.

Homotopy Homotopy is a continuous interpolation of paths (it will be clear later what we mean by path) in a topological space, that is, a transformation which smoothly connects paths. Thus, homotopy can be seen as a relation between paths and therefore it divides topological spaces in equivalence classes of all the paths that are related by a homotopy [17](*i.e.* that are continously deformable into each other, meaning that they are equivalent).
Homotopy classes of paths can be composed together with a product rule. Thus, it can be shown that a set of homotopies is a group, called a *homotopy group*. In fact we can define the product of two homotopy classes $[\alpha]$ and $[\beta]$ as $[\alpha] * [\beta] = [\alpha * \beta]$. It is demonstrated in [17] that the group properties are respected, that is

- Associativity: $([\alpha] * [\beta]) * [\gamma] = [\alpha] * ([\beta] * [\gamma])$
- Unit element: $[\alpha] * [I] = [\alpha]$ and $[I] * [\alpha] = [\alpha]$
- Inverse: $[\alpha] * [\alpha^{-1}] = [I]$, thus $[\alpha]^{-1} = [\alpha^{-1}]$

The most simple homotopy group that we can consider in a topological space is the homotopy group of loops, called the *fundamental group*. In this case the "paths" we referred to are loops, meaning any continuous function ϕ from $[0, 1]$ to the topological space X such that $\phi(0) = \phi(1) = x_0$ with x_0 called *base point*. The fundamental group can be defined as the homotopy group of continuous functions from the one dimensional sphere to the topological space X, namely:

$$\phi : S^1 \to X \tag{1.24}$$

In this sense the fundamental group is the group maps that continuously transform loops. That means that given two loops ϕ and ϕ' with the same base point there exist a smooth function $h(t)$ such that $h(0) = \phi$ and $h(1) = \phi'$. Basically, when considering loops a homotopy is just a deformation of one loop into another that does not break the loop at any point anytime.

In such case the loops are said to be homotopic and this relations splits the topological space X in equivalence classes for the fundamental group:

$$\pi_1(X) = \textit{loops}/\textit{homotopy} \tag{1.25}$$

which can be read as the set of all loops not connected by homotopy. A few examples can shed more light

- $\pi_1(S^2) = 0$. The topological space in consideration is the 2-dimensional sphere. Every loop on it can be reduced to a point so they are all connected by a homotopy, *i.e.* in the same equivalence class. Thus this group can be considered as the group made by only the neutral element 0.

- $\pi_1(S^1) = \mathbb{Z}$. On a circle, the loops that cannot be deformed into each other continuously are the ones with a different number of windings around the origin. The windings count can be positive or negative if turning clockwise or counter clockwise. Therefore the homotopy group is $\mathbb{Z}$.

- $\pi_1(\text{2D Torus}) = \mathbb{Z}^2$. The 2-dimensional torus is an extention of S^1. The loops winding on the vertical axis cannot be transformed into the loops winding around the horizontal axis, so we need at least two numbers to identify these homotopy classes. In addition to that the loops on both axis can wind more then once as if they were on a circle, thus both directions are characterized by $\mathbb{Z}$ giving us a $\mathbb{Z}^2$ class.

Generalizing from the fundamental group we define $\pi_n(X)$ the homotopy equivalence classes of maps from the n dimensional sphere to the topological space X:

$$\phi : S^n \to X \tag{1.26}$$

Ten-fold Classification In our context the maps are from the BZ to the target spaces defined by $Q(\boldsymbol{k})$ and by the different dimensions of the BZ. This means that the job of finding all the different topological classes is substituted by finding all the homotopy classes of the 10 categories for every dimensions. These results are taken from [17] and are shown in table 1.2 for different dimensions.
It was proven that the homotopy classes are periodic in the dimensions with a period of 8. Thus, all symmetry categories are classified for every dimension in the free fermionic Hamiltonians case.

The homotopy class represents the (simplified) Hamiltonians, with certain symmetries, which cannot be transformed into each other. That translates into the number of groundstates of those Hamiltonians that cannot be deformed into each other, in our case without closing the energy gap.

Class	$\mathcal{T}$	$\mathcal{C}$	$\mathcal{S}$	1	2	3	4	5	6	7	8
A	$\varnothing$	$\varnothing$	$\varnothing$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$
AIII	$\varnothing$	$\varnothing$	$+1$	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0	$\mathbb{Z}$	0
AII	-1	$\varnothing$	$\varnothing$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$\mathbb{Z}$
DIII	-1	$+1$	1	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$\mathbb{Z}$	0
D	$\varnothing$	$+1$	$\varnothing$	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0	$\mathbb{Z}$	0
BDI	$+1$	$+1$	1	$\mathbb{Z}$	0	0	0	$\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$
AI	$+1$	$\varnothing$	$\varnothing$	0	0	0	$\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$
CI	$+1$	-1	1	0	0	$\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0
C	$\varnothing$	-1	$\varnothing$	0	$\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0
CII	-1	-1	1	$\mathbb{Z}$	0	$\mathbb{Z}_2$	$\mathbb{Z}_2$	$\mathbb{Z}$	0	0	0

Table 1.2: Homotopy groups of the 10-fold classification, adapted from [2], obtained by Schnyder *et al.* [18]. The columns, in order from the left, show the Cartan labels, the absence($\varnothing$) or presence of a symmetry with its eigenvalue (±1) and the dimension of the space. The homotopy groups are periodic in the number of dimensions. For the first two "complex" classes the periodicity is modulo 2. While for the "real" classes the periodicity is modulo 8. They are called complex and real because the former do not have any reality condition on their first particle Hamiltonian, while the latter do [3].

As a concrete example let us consider the A class of the 10-fold classification which consists of Hamiltonians with no symmetries. Thus, the simplified $Q(\boldsymbol{k})$ can be transformed into a diagonal form by a Unitary transformation

$$U_{\boldsymbol{k}}Q(\boldsymbol{k})U_{\boldsymbol{k}}^{\dagger} = \begin{pmatrix} \mathbb{1} & \mathbb{0} \\ \mathbb{0} & -\mathbb{1} \end{pmatrix} \tag{1.27}$$

where U is a $(m+n)$ unitary matrix and not unique because any multiplication of U by a $U(m)$ or $U(n)$ matrix would still lead to (1.27).
Therefore, $Q(\boldsymbol{k})$ describes a map from the BZ to the target space $U(n+m)/[U(n)\times U(m)]$.
In $D=1$, the BZ is S^1 so the homotopy group is the fundamental group and it can be shown that it is trivial on $U(n+m)/[U(n)\times U(m)]$, meaning it is 0. In $D=2$ the

BZ is a 2D torus but can be extended out of a lattice for simplicity on a 2-sphere. Also $Q(\boldsymbol{k})$ can be seen to be a 2-sphere, giving us

$$\pi_2(S^2) = \mathbb{Z} \tag{1.28}$$

In fact the quantum Hall effect belongs to this class of symmetries and the integer characterizing each of its topological phases are the quantized Hall resistivities.
Another relevant example is given by the Kitaev chain [4], the model of interest of this thesis. This model possess the three classifying symmetries and thus belongs to class BDI. In one dimension, reading from table (1.2) we can infer that $\mathbb{Z}$ labels the different groundstates. That means we can associate an integer number to every groundstate. These numbers are the winding numbers around the origin of the map from the BZ to the target space which in this case is a two dimensional flat space.

1.2 Topological Invariants

Topology studies how systems can be transformed into each other and what are the elements that do not change in this transformation. In condensed matter physics we ask ourselves which Hamiltonians can be continuously transformed into each other and what they look like.
This concept was already introduced in the last section. We now give a more accurate description of what a topological phase is, how Hamiltonians can be transformed and what characterize the different phases beyond symmetry.
We can define an *adiabatic deformation* as a continuous transformation of a Hamiltonian in which

- the parameters are modified slowly and continuously
- the gap does not close
- the main symmetries are respected

and we can say that two Hamiltonians that are adiabatically connected are in the same topological phase. If the gap closes during the transformation of some parameters we will say that the system underwent a *topological phase transition.*
Therefore, the most intuitive task to do is to find properties of the quantum system that do not change during the adiabatic deformation. These are called *topological invariants.* Topological invariants are quantum numbers that can be (sometimes in

a difficult way) calculated for any systems and system with the same invariant are in the same topological phase.
Adiabatic transformations are the base element of the *adiabatic theorem*, a key concept in quantum mechanics which state that a physical system remains in its instantaneous eigenstate if a given perturbation is acting on it slowly enough and if there is a gap between the eigenvalue and the rest of the Hamiltonian's spectrum. The adiabatic theorem is of central importance in the study of the Berry phase and topological invariants.

The Berry Phase The main focus of the study of topological phases of matter is the ground state of a system. That is because we expect to see the topological behaviour of a model in it.
In his paper [19], M.V. Berry showed how a ground state changes when its Hamiltonian is slowly changed.
Let us consider an adiabatic deformation which is slow in a sense that the adiabatic theorem, stated above, is respected. If the system goes back to its original state at the end of the deformation, there will be two phases picked up by the state. One associated to the time evolution and one, called a Berry phase, with interesting properties.
Let us go into details, following Berry's work [19]. Consider an Hamiltonian depending on some parameters $\boldsymbol{R}$ which vary from time $t = 0$ to $t = T$. If $\boldsymbol{R}(0) = \boldsymbol{R}(T)$ the system follows a closed path $\mathcal{C}$ in the parameter space.
The system eigenstates $|\psi(t)\rangle$ will evolve according to

$$H(\boldsymbol{R})|\psi(t)\rangle = i\hbar|\dot{\psi(t)}\rangle \tag{1.29}$$

and, given the adiabatic theorem, at any time we can consider an instantaneous orthonormal basis from the eigenstates of $H(\boldsymbol{R})$ as $|n(\boldsymbol{R})\rangle$ (with $\boldsymbol{R} = \boldsymbol{R}(t)$), for which

$$H(\boldsymbol{R})|n(t)\rangle = E_n(\boldsymbol{R})|n(t)\rangle \tag{1.30}$$

This basis is not unique, due to gauge-invariance of the states by any $\boldsymbol{R}$-dependent smooth function. So we can make a gauge choice requiring that the phase of the eigenbasis is smooth and single valued on the parameter path $\mathcal{C}$ [20]. A system prepared in a $|n(\boldsymbol{R}(0)\rangle$ eigenstate will evolve into a state $|n(\boldsymbol{R}(T))\rangle$ so that the final state can be written as

$$|\psi(t)\rangle = e^{-\frac{i}{\hbar}\int_0^t dt' E_n(\boldsymbol{R}(t'))} e^{i\gamma_n(t)} |n(\boldsymbol{R}(t))\rangle \tag{1.31}$$

While the common time-evolution phase can be erased by a gauge transformation, the second phase $e^{i\gamma_n(t)}$ cannot. It is called the *Berry Phase*.
Plugging equation (1.31) into (1.29) we get

$$\dot{\gamma_n}(t) = i\langle n(\boldsymbol{R})|\nabla_{\boldsymbol{R}}|n(\boldsymbol{R})\rangle \dot{\boldsymbol{R}} \tag{1.32}$$

so that if we integrate this last relation we obtain an expression for the berry phase

$$\gamma_n(\mathcal{C}) = i\int_{\mathcal{C}} \langle n(\boldsymbol{R})|\nabla_{\boldsymbol{R}}|n(\boldsymbol{R})\rangle d\boldsymbol{R} \tag{1.33}$$

as a path integral on the parameter space which does not depend on the rate of change of the parameters.
From (1.33) we can define a vector-valued function

$$\gamma_n = \int_{\mathcal{C}} d\boldsymbol{R}\cdot\mathcal{A}_n(\boldsymbol{R}) \qquad\qquad \mathcal{A}_n(\boldsymbol{R}) = i\langle n(\boldsymbol{R})|\nabla_{\boldsymbol{R}}|n(\boldsymbol{R})\rangle \tag{1.34}$$

The vector $\mathcal{A}_n(\boldsymbol{R})$ is called the *Berry connection* or *Berry vector potential* in analogy to the gauge theory of vector potentials.
In fact we can show that the Berry connection is gauge-dependent. Making a parameter dependent gauge transformation as

$$|n(\boldsymbol{R})\rangle \to e^{\zeta(\boldsymbol{R})}|n(\boldsymbol{R})\rangle \tag{1.35}$$

with $\zeta(\boldsymbol{R})$ an arbitrary continuous function, $\mathcal{A}_n(\boldsymbol{R})$ transforms as

$$\mathcal{A}_n(\boldsymbol{R}) \to \mathcal{A}_n(\boldsymbol{R}) - \nabla_{\boldsymbol{R}}\zeta(\boldsymbol{R}) \tag{1.36}$$

As a consequence the Berry phase (1.33) will be modified by $\zeta(\boldsymbol{R}(0)) - \zeta(\boldsymbol{R}(T))$. Before Berry's article, the Berry phase was considered trivial because thanks to the gauge redundance $\zeta(\boldsymbol{R})$ could be chosen so to cancel out the contribution of γ_n [20]. Berry's idea to consider a closed path on $\mathcal{C}$ cast light again on this issue. Since we asked the phase to be single valued the gauge function must respect

$$\zeta(\boldsymbol{R}(0)) - \zeta(\boldsymbol{R}(T)) = 2\pi \times m \tag{1.37}$$

with m an integer, so that the phase γ_n can only be changed by an integer multiple of 2π and cannot be removed. Thus, on a closed path the Berry phase is gauge invariant, making it a physical observable quantity, in general given by

$$\gamma_n = \oint_{\mathcal{C}} d\boldsymbol{R} \cdot \mathcal{A}(\boldsymbol{R}) \tag{1.38}$$

This definition underlines both the independency of the phase by how the parameters change in time and the dependency on the geometry of the chosen loop.
Following the analogy to electrodynamics we define a gauge field tensor from the Berry connection

$$\Omega^n_{\mu\nu}(\boldsymbol{R}) = \partial_\mu \mathcal{A}^n_\nu(\boldsymbol{R}) - \partial_\nu \mathcal{A}^n_\mu(\boldsymbol{R}) \tag{1.39}$$

where $\partial_\mu = \frac{\partial}{\partial R^\mu}$. It is called *Berry's Curvature*, a gauge invariant field which allows us to define γ_n as a surface integral

$$\gamma_n = \int_{\mathcal{S}} dR^\mu \wedge dR^\nu \frac{1}{2} \Omega^n_{\mu\nu}(\boldsymbol{R}) \tag{1.40}$$

where we used the Stokes' theorem and $\mathcal{S}$ is any surface enclosed by the path $\mathcal{C}$.
In three dimensional space the Berry curvature (1.39) obtains the familiar form of the magnetic field of a vector potential, while the Berry phase becomes its surface integral. That is,

$$\boldsymbol{\Omega_n}(\boldsymbol{R}) = \nabla_{\boldsymbol{R}} \times \boldsymbol{\mathcal{A}}(\boldsymbol{R}) \qquad\qquad \gamma_n = \int_{\mathcal{S}} d\boldsymbol{S} \cdot \Omega_n(\boldsymbol{R}) \tag{1.41}$$

In his paper [21], Berry showed also how to switch the derivatives on the states, which can be computationally cumbersome, to derivatives of the Hamiltonian. Using

$$\langle n|\partial_\mu|m\rangle = \frac{\langle n|\partial_\mu H|n\rangle}{E_n - E_m} \tag{1.42}$$

and the completeness relation, we obtain

$$\Omega^n_{\mu\nu}(\boldsymbol{R}) = i \sum_{m\neq n} \frac{\langle n|\partial_\mu H|m\rangle\langle m|\partial_\nu H|n\rangle - (\mu \longleftrightarrow \nu)}{(E_n - E_m)^2} \tag{1.43}$$

which has the advantage of not having any differentiation on the wavefunction, so it can be evaluated for any gauge.
Using the adiabatic approximation we have focused on the evolution of one energy state. Equation (1.43) then looks like an interaction with the left out energy levels. In fact, including all the energy levels in the calculation adds up to 0:

$$\sum_n \Omega^n_{\mu\nu}(\boldsymbol{R}) = 0 \tag{1.44}$$

which is a local conservation law for the Berry curvature. In addition to that, equation (1.43) is singular wherever two energy levels energies coincide on a point $\boldsymbol{R}$ of the parameter space. This point corresponds to a monopole in parameter space.

Chern and Winding Numbers One of the mathematical theorems that links geometry to topology is the Gauss Bonnet theorem. It is applied to 2D compact Riemannian manifolds without boundary and it connects the Gaussian curvature of the manifolds to its number of genuses:

$$2(1-g) = \frac{1}{2\pi}\int_{\mathcal{M}} d^2\boldsymbol{x} F(\boldsymbol{x}) \tag{1.45}$$

where g is the genus, meaning the number of holes of the manifold $\mathcal{M}$ have, and $F(\boldsymbol{x})$ is the curvature, a local feature of a manifold.
The Gaussian curvature can be seen as the angle mismatch of a tangent vector after it was parallel transported around an infinitesimal closed loop on $\mathcal{M}$ [2].
In the physical systems we consider, the manifold $\mathcal{M}$ is always the Brillouin Zone and its tangent plane is spanned by the Bloch states of an occupied band at given momentum $\boldsymbol{k}$. The generalization to this context of the Gaussian curvature is the Berry curvature.
The generalization of the Gauss-Bonnet theorem in algebraic topology is the quantity

$$2\mathcal{C} = \frac{i}{2\pi}\int_{BZ} \operatorname{Tr} F = 2\frac{i}{2\pi}\int_{BZ} d^2\boldsymbol{k} \operatorname{Tr} F_{12} \tag{1.46}$$

known as the first Chern number. It is a gauge-invariant quantity which can only take integer values. It can be generalized to any $d = 2s$ space using the s-th power of the local Berry curvature F and taking its trace so to still have a gauge-invariant quantity.

We can see from table (1.2) that classes without chiral symmetry have an integer $\mathbb{Z}$ topological invariant when the dimension of the space is even, in that case we can consider the s-th Chern number as the topological invariant.
Let us now consider classes with chiral symmetry $\mathcal{C}$, which are the systems that we will be dealing with throughout this work.
We want to create an extension of the Chern number. In order to do so we consider the flatband Hamiltonians $Q(k)$ mentioned in the last section and we define

$$C^s = \varepsilon_{i_1 \dots i_d} \int_{BZ} \mathrm{d}^d\boldsymbol{k}\, \mathrm{Tr}[Q(\boldsymbol{k})\partial_{i_1} Q(\boldsymbol{k}) \dots \partial_{i_d} Q(\boldsymbol{k})] \tag{1.47}$$

for $d = 2s$. This new topological invariant can be seen as the *winding number* of the unitary transformation $Q(\boldsymbol{k})$ over the compact BZ. It can be shown by calculations that $C^s = 0$ for all odd spatial dimensions.
In odd dimension spaces we can define a different topological winding number, that is

$$W^{(s)} = \frac{(-1)^s s!}{2(2s+1)!}\left(\frac{i}{2\pi}\right)^{s+1} \varepsilon_{i_1 \dots i_d} \int_{BZ} \mathrm{d}^d\boldsymbol{k}\, \mathrm{Tr}[\mathcal{C}Q(\boldsymbol{k})\partial_{i_1} Q(\boldsymbol{k}) \dots \partial_{i_d} Q(\boldsymbol{k})] \tag{1.48}$$

So that it can be interpreted as the winding number of the off-diagonal part of the flatband Hamiltonian.
With the Chern and winding numbers we have a flag that represents the different topological sectors of any symmetry class for systems with translational invariance.
It is important for our work to state that if interactions which do not break the defining symmetry (neither spontaneously nor explicitely) are included in the picture two things may happen

- Phases with a different winding number as defined above may be adiabatically connected to each other

- The interaction changes the symmetry classification of topological phases with topological characteristics different from the ones in the non interacing phases

Interactions will also have an impact on the conditions of existence of edge modes in the model.

1.3 The Kitaev Model

The main focus of this thesis is to work on the Kitaev Model [4], developed by A. Y. Kitaev while studying fault-tolerant quantum computation.
Kitaev proposed a one dimensional superconducting model, a "quantum wire". as a solution to the quantum errors that occur when dealing with quantum computation in order to create a valid *quantum memory.*
There are two kinds of error that quantum systems are sensitive to: a classical and a phase error. The former is due to a σ_j^x operator that can flip the state of the j-th qubit, the latter is due to the σ_j^z operator that flips the sign of all the states with the jth qubit equal to 1 with respect to all the states where that qubit is equal to 0. Although these errors can be corrected it is usually hard to work on both of them. This is where Kitaev introduced his model.
The idea is simple: firstly we are in an electronic system. Here the electrons can work as qubits if we consider an empty state as $|0\rangle$ and an occupied state as $|1\rangle$. In this picture a classical error would mean to destroy or create an electron which is not allowed by the conservation of electronic charge (same would happen in a superconductor where parity is conserved). Yet, if an electron could jump from one site to another we would have two state flips, meaning two classical error. So this must be avoided placing the fermions far apart enough.
The phase error in this context would be represented by the number operator and it would not be lifted. The solution to this problem is to introduce the Majorana fermions.

1.3.1 Majorana Modes

Majorana particles are interesting and important because they can be seen as building blocks of fermions. Specifically, a fermion can be decomposed into two Majorana quasi-particles because of their particular statistics. Let us consider N fermionic creation/annihilation operators $c_1, \ldots, c_N$ and $c_1^\dagger, \ldots, c_N^\dagger$ with the usual statistics:

$$\{c_i, c_j\} = \{c_i^\dagger, c_j^\dagger\} = 0 \qquad\qquad \{c_i^\dagger, c_j\} = \delta_{ij} \tag{1.49}$$

where i, j run on the different N sites. Let us split the fermionic operators into a real and imaginary part

$$c_i = \frac{1}{2}(\gamma_{2i-1} + i\gamma_{2i}) \qquad\qquad c_i^\dagger = \frac{1}{2}(\gamma_{2i-1} - i\gamma_{2i}) \tag{1.50}$$

and call γ_{2j-1} and γ_{2j} Majorana fermions. By inverting this formula and imposing the fermionic anti-commutation rules (1.49) we obtain a definition for the Majorana fermions

$$\gamma_{2i-1} = c_i + c_i^\dagger \qquad\qquad \gamma_{2i} = -i(c_i - ic_i^\dagger) \tag{1.51}$$

and their anti-commutation relation

$$\{\gamma_i, \gamma_j\} = 2\delta_{ij} \tag{1.52}$$

First of all, we obtain that $\gamma_i^\dagger = \gamma_i$. Second of all, their exchange relation is very interesting, it imposes that $\gamma_i^2 = 1$ which means that Majorana fermions do not respect the Pauli Exclusion Principle. In addition to that, if we created a number operator we would get $\gamma_i^\dagger\gamma_i = \gamma_i^2 = 1$ so there is no real meaning of counting Majoranas.
The idea of generating Majorana fermions as the real and imaginary part of a fermion seems just a mathematical operation because Majoranas are quasi-particles that are created together and cannot be spatially separated. There is, however, some physical insight: we can see a fermion as a superposition of two Majorana fermions states that *can* be spatially separated. These states would not suffer from the phase error that typically affects fermionic states. In fact, as mentioned above, the phase error is given by the number operator $c_i^\dagger c_i$ which is unlikely to accur if the Majorana modes belonged to different sites. This is the starting point for Kitaev to look for a model that has Majorana fermions as low energy degrees of freedom.

1.3.2 The Model

The nature of Majorana fermions makes them be their own anti-particle, or in a more "particle-hole" language their own "hole". Thus, we can expect such fermions to be an equal superposition of an electron and a hole state [22]. This idea suggests to look for Majorana fermions in superconducting systems because in the Bogoliubov-de Gennes Hamiltonian formalism we work with Bogoliubov quasi-particles which

are indeed superpositions of electron and hole states. Another reason as why we should consider a superconducting system is that it breaks down the gauge symmetry of normal fermionic operators (*i.e.* the system invariance for transformations like $c_i \to e^{i\phi}c_i$) to $\mathbb{Z}_2$ so that Majorana operators do not mix with other operators.
In the most simple model we can create we consider a chain of N superconducting fermions, with fixed spin, which gives us the Kitaev Hamiltonian

$$H = \sum_i \Big[-t\Big(c_i c^\dagger_{i+1} + c_{i+1} c^\dagger_i \Big) - \mu \Big(c^\dagger_i c_i - \frac{1}{2}\Big) + \Delta \Big(c_i c_{i+1} + c^\dagger_i c^\dagger_{i+1}\Big)\Big] \tag{1.53}$$

where the index i runs on all the N sites, t is the *hopping parameter*, μ the chemical potential and $\Delta = |\Delta| e^{i\theta}$ the superconducting gap (whose phase parameter can be absorbed into the fermionic operators so that $|\Delta| = \Delta$).
Rewriting the Kitaev Hamiltonian using (1.50) we obtain

$$H_\gamma = \frac{i}{2}\sum_i \big[-\mu(\gamma_{2i-1}\gamma_{2i}) + (t + |\Delta|)\gamma_{2i}\gamma_{2i+1} + (-t + |\Delta)\gamma_{2i-1}\gamma_{2i+2}\big] \tag{1.54}$$

In this form it is easier to show the appearence of isolated Majorana modes for some values of the parameters. Let us consider two cases:

- $|\Delta| = t = 0, \mu < 0$. This brings (1.54) to the form

$$H_\gamma = -\mu \sum_i \Big(c^\dagger_i c_i - \frac{1}{2}\Big) = -\frac{\mu}{2}\sum_i \Big(\gamma_{2i-1}\gamma_{2i}\Big) \tag{1.55}$$

  and the Majorana operators γ_{2i-1} and γ_{2i} of the same site are paired together in a way to form a groundstate with a vanishing occupancy number. This is called the *trivial phase.* This configuration is graphically shown in the top part of Fig. 1.1.

- $|\Delta| = t > 0, \mu = 0$. With this parameters we have

$$H_\gamma = it \sum \gamma_{2i}\gamma_{2i+1} \tag{1.56}$$

  which clearly shows that Majoarna operators of different sites are paired with each other (see bottom part of Fig. 1.1). This is called the *topological phase.* This suggests to consider creation/annihilation operators that spread over two consecutive sites as $\tilde{c}_i = \frac{1}{2}(\gamma_{2i} + i\gamma_{2i+1})$, $\tilde{c^\dagger_i} = \frac{1}{2}(\gamma_{2i} - i\gamma_{2i+1})$ so to create a Hamiltonian in the form $H = 2t\sum_{i=1}^{N-1}(\tilde{c}^\dagger_i \tilde{c}_i - \frac{1}{2})$. Its groundstate is defined

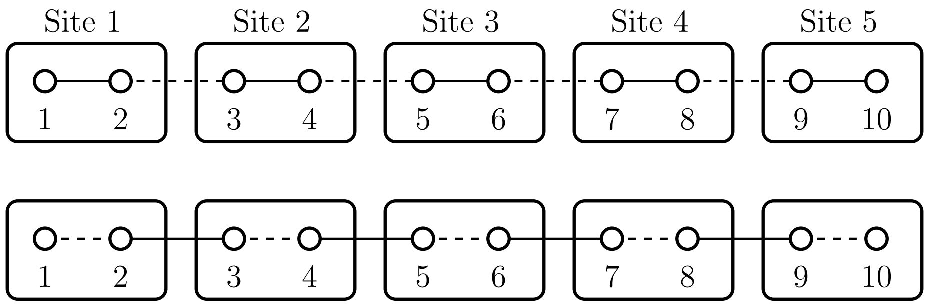


Figure 1.1: Graphical representation of the two possible couplings for the Majorana fermions. The pictures show the fermionic sites of the Kitaev chain labeled by their number and represented by a box. The latter is bipartite into two Majorana modes. The chain at the top shows the coupling between Majoranas inside the same site, which leads to the trivial phase. The chain at the bottom shows the coupling between Majorana of different sides which shows the topological behaviour. See how the first and last Majorana fermions are decoupled from the chain.

from $\tilde{c}_i|\psi\rangle = 0$ for $i = 1, \ldots, N$. So it does not contain any $\tilde{c}_i$ particle but there are two Majorana modes that can host zero-energy excitations because the operators γ_1 and γ_{2N} do not show up in the Hamiltonian (1.56). Two orthogonal states $|\psi_0\rangle$, $|\psi_1\rangle$ respect the vacuum condition and we can see how they differ in parity.
We can define a fermionic parity operator as

$$P = \prod_i \Big(- i\gamma_{2i-1}\gamma_{2i} \Big) = \prod_i \Big(1 - 2c_i^\dagger c_i \Big) \tag{1.57}$$

which is +1/-1 when the system has an even/odd number of fermions. To act on our groundstates let us define a fermionic non local operator $\Psi = \frac{1}{2}(\gamma_{2N} + i\gamma_1)$ so that the parity operator reduces to $P = 1 - 2\Psi^\dagger\Psi = -i\gamma_1\gamma_{2N}$ and we can see its action on $|\psi_0\rangle$ and $|\psi_1\rangle$

$$-i\gamma_1\gamma_{2N}|\psi_0\rangle = |\psi_0\rangle \qquad\qquad -i\gamma_i\gamma_{2N}|\psi_1\rangle = -|\psi_1\rangle \tag{1.58}$$

This means that $|\psi_0\rangle$ does not host excitations of the Ψ particle while $|\psi_1\rangle$ does.

The two specific cases we just explored show two different behaviours of the Kitaev

Model. In the first one the coupling of Majorana pairs does not allow any edge modes, while the second one does, leading to a degeneracy in the groundstates: the one that contains a non local pair of Majorana fermions and the one that does not, simply because there is no energy cost in creating such non local Majorana couple. Even though we only worked with two specific sets of parameters the appearence of zero-energy Majorana modes is not only confined to the $\mu = 0$ case but to a whole interval of values of μ.

Energy Spectrum To get more insight into the existence of the Majorana modes let us solve the spectrum of the Kitaev Hamiltonian (1.53) in the Bogoliubov-de Gennes formalism [23], which we will only quote briefly.
Using the BdG method consists in doubling the degrees of freedom of our system and exploit this redundancy to write the Hamiltonian in a very convenient form, useful for diagonalization. The redundancy is due to working with combinations of creation/annihilation operators of particles and antiparticles, thus it also produces a symmetric energy spectrum with respect to the zero energy level.
The BdG approach requires us to impose Periodic Boundary Conditions (PBC) on the Kitaev chain. This prevents the generation of edge modes because there is no edge anymore. Anyhow, we now want to study the bulk properties and see if they can somehow show signs of the presence of the edge modes. Thus, there is no problem in imposing PBC.
In order to do so we write the fermionic operators in momentum space through a Fourier transformation

$$c_n = \frac{1}{\sqrt{N}} \sum_k e^{ikn} c_k \qquad\qquad c_n^\dagger = \frac{1}{\sqrt{N}} \sum_k e^{-ikn} c_k^\dagger \tag{1.59}$$

with k the discretized momenta $k = 2\pi i$ where $i = 0, \dots, N-1$, then the new Hamiltonian is

$$H = \sum_k \Big[-(t \cos k + \mu) c_k^\dagger c_k - i\Delta \sin k \big(c_{-k} c_k + c_k^\dagger c_{-k}^\dagger \big) \Big] \tag{1.60}$$

Written in the Bogoliubov-de Gennes formalism we get

$$H = \frac{1}{2} \sum_k \begin{pmatrix} c_k^\dagger & c_{-k} \end{pmatrix} \begin{pmatrix} -2t \cos k - \mu & i2\Delta \sin k \\ -i2\Delta \sin k & 2t \cos k \mu \end{pmatrix} \begin{pmatrix} c_k \\ c_k^\dagger \end{pmatrix} \tag{1.61}$$

So that it is easy now to find the energy spectrum

$$E(k) = \pm\sqrt{(2t\cos k + \mu)^2 + (2\Delta \sin k)^2} \tag{1.62}$$

which clearly shows a gap for all values $2|t| \neq |\mu|$ except for $2|t| = |\mu|$ where the gap closes for the values of $k = \pm\pi$. This means that we can expect a phase transition at $\mu = \pm 2t$ and that is exactly the point that discriminates the topological phase with Majorana edge modes to the one without.

Majorana zero energy modes at the boundary In the last paragraph we showed how two different kinds of coupling realized on the Kitaev chain produce a trivial and a topological phase, defined by the absence or presence of Majorana modes at the boundaries. The study of the bulk properties - the energy spectrum - shows that the two phases are separated at the critical points $|\mu| = 2|t|$.
Starting from this hypothesis we want to explicitely calculate the Majorana modes wavefunction and show how they are localized at the edges of the chain as it is done in [24].
Let us redefine for clarity the Majorana fermions in this way

$$\gamma_{L,j} = \gamma_{2j-1}, \qquad\qquad \gamma_{R,j} = \gamma_{2j}. \tag{1.63}$$

Considering $\Delta = t = 1$ the Hamiltonian for $\mu = 0$ is

$$H = i\sum_{j=1}^{N-1} \gamma_{R,j}\gamma_{L,2j+1} \tag{1.64}$$

and for $\mu \neq 0$ is

$$H = -\frac{i\mu}{2}\sum_{j}^{N} \gamma_{L,j}\gamma_{R,j} + i\sum_{j=1}^{N-1} \gamma_{R,j}\gamma_{L,j+1}. \tag{1.65}$$

The latter Hamiltonian (1.65) implies that the Majoranas at the far right and left end of the chain are uncoupled. We will name them $\Psi_L = \gamma_{L,1}$ and $\Psi_R = \gamma_{R,N}$. In order to show how these two states are localized at the edges we consider an ansatz for their wavefunction in this form

$$\Psi_L = \sum_{j=1}^{N} \alpha_j \gamma_{L,j}, \qquad\qquad \Psi_R = \sum_{j=1}^{N} \beta_j \gamma_{R,j}. \tag{1.66}$$

Let us start with Ψ_L, if it represent a zero energy mode it must satisfy the relation

$$[H, \Psi_L] = 0, \tag{1.67}$$

which gives us

$$i\mu \sum_{j=1}^{N} \alpha_j \gamma_{R,j} - i \sum_{j=1}^{N-1} \alpha_{j+1} \gamma_{R,j} = 0. \tag{1.68}$$

This can be rewritten so to isolate the $\gamma_{R,j}$, namely:

$$\sum_{j=1}^{N-1} \Big(-\frac{\mu}{2}\alpha_j + \alpha_{j+1} \Big) \gamma_{R,j} - \frac{\mu}{2}\alpha_N \gamma_{R,N} = 0. \tag{1.69}$$

We can solve this setting all the coefficients multiplying $\gamma_{R,j}$ to 0 and find a recursive relation so that α_N is completely determined. That implies:

$$-\frac{\mu}{2}\alpha_j + \alpha_{j+1} = 0, \tag{1.70}$$

which leads to

$$\alpha_j = \left(\frac{\mu}{2}\right)^{j-1} \tag{1.71}$$

by asking that $\Psi_L(\mu = 0) = m_{L,1}$, which sets $\alpha_1 = 1$. Now the complete wavefunction for the left Majorana edge state is

$$\Psi_L = \sum_{j=1}^{N} \left(\frac{\mu}{2}\right)^{j-1} \gamma_{L,j} \tag{1.72}$$

which, thanks to the exponential term, is localized at the edge of the chain for $\mu < 2$. This is in accordance with the condition $\mu < 2t$ we have for the existence of this mode.
The same steps can be repeated for the right Majorana mode leading to

$$\Psi_R = \sum_{j=1}^{N} \left(\frac{\mu}{2}\right)^{N-j} \gamma_{R,j}. \tag{1.73}$$

While both states Ψ_L and Ψ_R are localized at the (opposite) edges, none of them commutes with the Hamiltonian if not on the thermodynamic limit $N \to \infty$, that is

$$[H, \Psi_L] = i\frac{\mu}{2}\left(\frac{\mu}{2}\right)^{N-1}\gamma_{R,N}, \qquad [H, \Psi_R] = -i\frac{\mu}{2}\left(\frac{\mu}{2}\right)^{N-1}\gamma_{L,1} \tag{1.74}$$

The two modes can be composed into one non-local fermion as $\Psi = \frac{1}{2}(\Psi_L + i\Psi_R)$ which still does not commute with the Hamiltonians in the finite size limit but it has the exponentially decaying term in μ. Thus for $N \to \infty$ the non-local fermion has zero energy and the right and left Majoranas become unpaired.

1.3.3 Correlation Functions

We now turn our attention to the determination of the correlation functions. They are the main tool to investigate the properties of the bulk and are the typical observable we measure in experiments. Specifically, the whole work on this thesis is based on studying the correlation of Kitaev chains in different interacting and non-interacting contexts.
In order to obtain the correlation functions we need to introduce the method developed by Lieb, Schultz and Mattis (LSM) [25] to put in a diagonal form our Kitaev Hamiltonian (1.53).
The LSM was intended for all the Hamiltonians of a lattice system of N sites that can be written in the form

$$H = \sum_{i,j}^{N}\left(c_i^\dagger A_{ij} c_j + \frac{1}{2}\left(c_i^\dagger B_{ij} c_j^\dagger + h.c.\right)\right) \tag{1.75}$$

where A and B are real matrix with A symmetric and B anti-symmetric. The difference with the Bogoliubov transformation is that there is no need to impose periodic boundary conditions or to do a Fourier transformation. In addition to that, there is no doubling of degrees of freedom and no double spectrum symmetric to the zero energy level.
The purpose of the LSM approach is to put any Hamiltonian of the form (1.75) in a diagonal form

$$H = \sum_k \Lambda_k \eta_k^\dagger \eta_k + cost \tag{1.76}$$

through a suitable linear transformation of the fermionic operators $c_i^\dagger$ and c_i:

$$\eta_k = \sum_k \left(g_{ki}c_i + h_k i c_i^\dagger\right) \qquad \eta_k^\dagger = \sum_k \left(g_k i c_i^\dagger + h_k i c_i\right) \tag{1.77}$$

with g_{ki} and h_{ki} real matrices.
We can now impose the following relation:

$$[\eta_k, H] - \Lambda_k \eta_k = 0 \tag{1.78}$$

and we will see later on how it is fundamental to impose the η_k to be canonical in their commutation relations. From this last relation we get the condition (we are following [26])

$$\begin{aligned} \sum_j g_{kj}A_{ji} - h_{kj}B_{ji} &= \Lambda_k g_{ki} \\ \sum_j g_{kj}B_{ji} - h_{kj}A_{ji} &= \Lambda_k h_{ki} \end{aligned} \tag{1.79}$$

which is true only if A and B are real. We define another set of matrices to simplify calculations, namely:

$$\begin{aligned} \phi_{ki} &= g_{ki} + h_{ki} \\ \psi_{ki} &= g_{ki} - h_{ki} \end{aligned} \tag{1.80}$$

So that writing for semplicity $\phi_k = \phi_{ki}$ and $\psi_k = \psi_{ki}$, equations (1.79) become

$$\begin{aligned} \phi_k(A-B) &= \Lambda\psi_k \\ \psi_k(A+B) &= \Lambda\phi_k \end{aligned} \tag{1.81}$$

That can be easily decoupled into

$$\begin{aligned} \phi_k(A-B)(A+B) &= \Lambda^2\phi_k \\ \psi_k(A+B)(A-B) &= \Lambda^2\psi_k \end{aligned} \tag{1.82}$$

These two pairs of equations (1.81) and (1.82) are the core of the LSM method, if calculate them we have an expression for the matrices ϕ_k and ψ_k which allow us to do the operator transformation that puts the original Hamiltonian (1.75) in the diagonal form (1.76).
Furthermore, since for the matrices the trivial relations $(A+B)^T = (A-B)$ holds, we can infer that the eigenvalues Λ_k^2 are real and positive and that ϕ_k and ψ_k are real and orthonormal. In particular, the orthonormality of the eigenvectors implies

that the η_k operators follow canonical anti-commutation relations (it can be show by easy computation).

$$\begin{aligned} \phi_k\phi_l &= \delta_{kl} \\ \psi_k\psi_l &= \delta_{kl} \end{aligned} \quad \Longleftrightarrow \quad \begin{aligned} \{\eta_k^\dagger, \eta_l\} &= \delta_{kl} \\ \{\eta_k, \eta_l\} &= 0 \end{aligned} \tag{1.83}$$

In the special case of the Kitaev Hamiltonian (1.53) we follow the steps of [26] and [27]. It is easy to see that the symmetric part is given by the hopping term and the chemical potential part while the superconducting side is antisimmetric by construction. Both matrices A, B representing these sectors are real.
We can see that these matrices are constant on the diagonals (the central one is referred to as 0 diagonal and then we count $+1$ or -1 going upwards/downwards). This is a property of the so called Toeplitz matrices, *i.e.* $A_{ij} = A_{i-j}$ in the sense that all the elements belongin to the same diagonal (i-j) are identical.
Specifically, the symmetric matrice A has a term $-\mu$ on the $i-j=0$ and a $-t$ term on the $i-j=\pm 1$ diagonals for symmetry reasons. The B matrix will have the Δ term on both the $+1$ an -1 diagonals but with opposite signs.
Since we are interesed in studying the properties of the bulk we can impose PBC which makes the A and B matrices even more peculiar. They are indeed *circulant matrices*, a special case of a Toeplitz matrix where $A_{ij} = A_{i-j \mod L}$ for some L.
Following the definition of a Toeplitz matrix we express the matrices A, B:

$$\begin{aligned} A_{ij} &= a(i-j) = -t(\delta_{i,i+1} + \delta_{i,i-1}) - \mu\delta_{i,j} \\ B_{ij} &= b(i-j) = \Delta(\delta_{i,i+1} - \delta_{i,i-1}) \end{aligned} \tag{1.84}$$

We recall now the main equations of the LSM method (1.81) and (1.82) derived previously. We do notice, though, that being the squared eigenvalues Λ_k^2 positive real numbers we can take them as a squared module *i.e* $|\Lambda_k|^2$. In addition to that we transpose the second pair of equations (1.82) (just a flip of sign for the B matrix) so they are presented in a more ordered fashion. All in all, we obtain

$$\phi_k(A+B)(A-B) = |\Lambda|^2\phi_k \qquad (A+B)\phi_k = |\Lambda|\psi_k \tag{1.85}$$

$$\psi_k(A-B)(A+B) = |\Lambda|^2\psi_k \qquad (A-B)\psi_k = |\Lambda|\phi_k \tag{1.86}$$

To find a solution to this set of equations we rely on the circulant nature of the matrices. In particular if A and B are circulant matrices so are $(A+B)$ and $(A-B)$

and their product too.
A theorem for circulant matrices states that any $N \times N$ circulant matrix M has eigenvectors

$$\psi_k = \frac{1}{\sqrt{N}}\big(1, e^{-i2\pi k/N}, e^{-i4\pi k/N}, \dots, e^{-i2\pi(n-1)k/N}\big)^T \tag{1.87}$$

for $k = 0, \dots, N-1$ and their corresponding eigenvalues

$$\lambda_k = \sum_j a_j e^{-i2\pi kj/N} \tag{1.88}$$

and the matrix can be diagonalized by the *Fourier matrix* F_N

$$(F_N)jk = \frac{e^{i2\pi kl/N}}{\sqrt{N}} \tag{1.89}$$

meaning that $M = F_N[\text{diag}(\lambda_k)]F_N^\dagger$.
To begin with, let us see that $(A-B)(A+B)$ and $(A-B)$ are both circulant matrices which commute. So we can select as basis of eigenvectors the normalised functions

$$\psi_{kj} = \frac{e^{i2\pi kj}}{\sqrt{N}} \tag{1.90}$$

which diagonalize both $(A-B)(A+B)$ and $(A-B)$. In particular using (1.86) we have

$$(A-B)\psi_k = \lambda_k\psi_k = |\Lambda_k|\phi_k \tag{1.91}$$

with λ_k is the eigenvalue of ψ_k with respect to the matrix (A-B). From this last relation we find that $\phi_k = \lambda_k\psi_k/|\Lambda_k|$ and since ϕ_k and ψ_k are normalized vectors, the quantity $\lambda_k/|\Lambda_k|$ must be a modulo one number, so we can directly set $\lambda_k = \Lambda_k$. We only need the eigenvalues now that can be expressed through the formula (1.88) to obtain

$$\Lambda_k = \sum_{l=-(N-1)/2}^{(N-1)/2} \big(a(l) - b(l)\big)e^{ikl} \qquad Neven \tag{1.92}$$

$$\Lambda_k = \sum_{l=-(N-1)/2}^{(N-1)/2} \big(a(l) - b(l)\big)e^{ikl} + (-1)^l a(N/2) \qquad Nodd \tag{1.93}$$

with k the discretized wave-number so that $k = 2\pi i/N$ with $i = 0, \dots, N-1$. The values of the eigenvalues we were looking for are thus the module of Λ_k.

Correlation Functions Calculation We now have all the ingredients to calculate the quantum correlation functions. We are mainly interested in one-body and two body correlations functions, namely

$$\langle c_i^\dagger c_j \rangle = \langle GS|c_i^\dagger c_j|GS\rangle \tag{1.94}$$

$$\langle c_i^\dagger c_j^\dagger \rangle = \langle GS|c_i^\dagger c_j^\dagger|GS\rangle \tag{1.95}$$

where (1.95) is called the anomalous correlator. The ground state is definied as the state annihilated by all the annihilation operators of the LSM quasiparticles $\eta_k|GS\rangle = 0$ for all k's.
In order to make this calculation we still follow the steps of [27] and [26]. An important characteristic of this approach is that it allows us to calculate the fermionic correlation functions starting from the expectation values of Majorana operators.
Let us show how we can do that. We start by recalling the relations between fermionic and Majorana operators (1.50) and (1.52)

$$c_i = \frac{1}{2}(\gamma_{2i-1} + i\gamma_{2i}) \qquad c_i^\dagger = \frac{1}{2}(\gamma_{2i-1} - i\gamma_{2i}) \tag{1.96}$$

$$\gamma_{2i-1} = c_i + c_i^\dagger \qquad \gamma_{2i} = -i(c_i - ic_i^\dagger) \tag{1.97}$$

and let us substitute them into the η_k operators like so

$$\begin{aligned}
\eta_k &= \sum_i \left(g_{ki}c_i + h_{ki}c_i^\dagger\right) \\
&= \sum_i \big(\frac{\phi_{ki} + \psi_{ki}}{2}c_i + \frac{\psi_{ki} - \psi_{ki}}{2}c_i^\dagger\big) \\
&= \sum_i \big(\phi_{ki}\frac{c_i + c_i^\dagger}{2} + \psi_{ki}\frac{c_i - c_i^\dagger}{2}\big)
\end{aligned}$$

So that using (1.97) we obtain

$$\begin{aligned}\eta_k &= \frac{1}{2}\sum_i \big(\phi_{ki}\gamma_{2i+1} - i\phi_{ki}\gamma_{2i}\big)\\ \eta_k^\dagger &= \frac{1}{2}\sum_i \big(\phi_{ki}\gamma_{2i+1} + i\phi_{ki}\gamma_{2i}\big)\end{aligned} \tag{1.98}$$

The last step is to invert these expressions using the orthonormality of the eigenfunctions ϕ_k and ψ_k

$$\gamma_{2i} = i\sum_k \psi_{ki}\big(\eta_k - \eta_k^\dagger\big) \qquad\qquad \gamma_{2i+1} = i\sum_k \phi_{ki}\big(\eta_k + \eta_k^\dagger\big) \tag{1.99}$$

so that now we can evaluate correlations functions for Majorana fermions.
Let us recall that the groundstate is the η_k vacuum. We want to calculate the groundstate expection value $\langle m_i m_j\rangle$ for all combinations of i,j even and odd.

$$\begin{aligned}\langle GS|\gamma_{2i+1}\gamma_{2j+1}|GS\rangle &= \sum_{mn}\phi_{mi}\phi_{nj}\langle GS|(\eta_m+\eta_m^\dagger)(\eta_n+\eta_n^\dagger)|GS\rangle\\ &= \sum_{mn}\phi_{mi}\phi_{nj}\langle GS|\eta_m\eta_n^\dagger|GS\rangle\\ &= \sum_m \phi_{mi}\phi_{mj} = \delta_{ij}\end{aligned}$$

where we used the orthonormality of the eigenfunctions ϕ_k, analogously we use the orthonormality of ψ_k to prove the following

$$\begin{aligned}\langle GS|\gamma_{2i}\gamma_{2j}|GS\rangle &= i^2\sum_{mn}\psi_{mi}\psi_{nj}\langle GS|(\eta_m-\eta_m^\dagger)(\eta_n-\eta_n^\dagger)|GS\rangle\\ &= \sum_{mn}\psi_{mi}\psi_{nj}\langle GS|\eta_m\eta_n^\dagger|GS\rangle\\ &= \sum_m \psi_{mi}\psi_{mj} = \delta_{ij}\end{aligned}$$

The less trivial expectation values are

$$\begin{aligned}\langle GS|\gamma_{2i}\gamma_{2j+1}|GS\rangle &= i\sum_{mn}\phi_{mi}\psi_{nj}\langle GS|(\eta_m-\eta_m^\dagger)(\eta_n+\eta_n^\dagger)|GS\rangle\\ &= i\sum_{mn}\psi hi_{mi}\psi_{nj}\langle GS|\eta_m\eta_n^\dagger|GS\rangle\\ &= i\sum_{m}\psi_{mi}\phi_{mj}\equiv -i(T_N)_{ij}\end{aligned}$$

and

$$\begin{aligned}\langle GS|\gamma_{2i+1}\gamma_{2j}|GS\rangle &= -i\sum_{mn}\phi_{mi}\psi_{nj}\langle GS|(\eta_m-\eta_m^\dagger)(\eta_n+\eta_n^\dagger)|GS\rangle\\ &= -i\sum_{mn}\psi_{mi}\psi_{nj}\langle GS|\eta_m\eta_n^\dagger|GS\rangle\\ &= -i\sum_{m}\psi_{mi}\phi_{mj}\equiv i(T_N)_{ij}\end{aligned}$$

We introduced the term T_N so to create a 2×2 off diagonal matrix in the following manner

$$C_{ij}=\begin{pmatrix}0 & (T_N)_{ij}\\ -(T_N)_{ji} & 0\end{pmatrix}\qquad\qquad (T_N)_{ij}=\sum_m\psi_{mi}\phi_{mj} \tag{1.100}$$

In order to write all the Majorana expectation values in a compact form

$$\langle GS|\gamma_i\gamma_j|GS\rangle=\langle\gamma_i\gamma_j\rangle=\delta_{ij}+i(C_N)_{ij} \tag{1.101}$$

where C_N is a $2N\times 2N$ block matrix with components $(C_N)_{ij}=(C_{ij})$ with $i,j=1,\dots,N$.
With these results we can calculate the fermionic correlation functions explicitely from the ones obtained with Majorana operators and the results of the spectrum calculations.
Let us consider the ϕ_k and ψ_k (1.90) equations we found. Since they are two complex functions, in order to calculate T_N we can consider the complex conjugate of ψ_k in place of ψ_k, so that the definition for T_N becomes

$$(T_N)_{jl}=\sum_{n=0}^{N-1}\bar{\psi}_{nj}\phi_{nl}=\frac{1}{2\pi}\sum_{k=0}^{2\pi(N-1)/N}\frac{\Lambda_k}{|\Lambda_k|}e^{-ik(j-l)} \tag{1.102}$$

which becomes, in the termodynamic limit

$$(T_N)_{jl} = \frac{1}{2\pi}\int_0^{2\pi} dk \frac{\Lambda_k}{|\Lambda_k|} e^{-ik(j-l)} \tag{1.103}$$

Now we have all the ingredients to calculate the fermionic correlation functions starting from

$$\begin{aligned}
\langle c_i^\dagger c_j \rangle &= \frac{1}{4}\langle(\gamma_{2i+1} + i\gamma_{2i})(\gamma_{2j+1} - i\gamma_{2j})\rangle & (1.104)\\
&= \frac{1}{4}(\langle\gamma_{2i+1} + i\gamma_{2j+1} - i\langle\gamma_{2i+1}\gamma_{2j}\rangle + i\langle\gamma_{2i}\gamma_{2j+1}\rangle + \langle\gamma_{2i}\gamma_{2j}\rangle) & (1.105)\\
&= \frac{1}{4}(2\delta_{ij} - (T_N)_{ji} - (T_N)_{ij}) & (1.106)
\end{aligned}$$

and similarly for the anomalous correlation functions

$$\langle c_i^\dagger c_j^\dagger \rangle = \frac{1}{4}(2\delta_{ij} + (T_N)_{ji} - (T_N)_{ij}) \tag{1.107}$$

1.4 Kitaev Model with Next Nearest Neighbor Coupling

In the last section we studied the Kitaev model in its original form, that is with nearest neighbor coupling. It is also interesting to show the effects of a longer range coupling, specifically the next to nearest neighbor (NNN) coupling. Considering this type of interaction was firstly proposed by Y. Niu et al [21] as a necessary step when dealing with the real implementation of a quantum wire. The model they applied it on was the quantum spin Ising chain which can be transformed into the Kitaev chain. So we will follow the work [24].
As mentioned above, the Kitaev Hamiltonian belongs to the BDI symmetry class which is characterized by a $\mathbb{Z}_2$ homotopy group. That means that different topological phases are characterized by a different integer number which is the amount of edge states. The Kitaev chain shows the presence of one edge state which is related to the winding number $\omega = \pm 1$. It is thus interesting to study models which can host a larger number of edge modes. The NNN Kitaev model is one of such.

Energy spectrum and phase diagram Let us start by writing the Kitaev Hamiltonian with NNN coupling. We define t_a and Δ_a as the *hopping paramter* and the *superconducting gap* of respectively, the Nearest Neighbor for $a = 1$ and the NNN interaction for $a = 2$. Let us set $t_1 = \Delta_1 = \lambda_1$ and $t_2 = \Delta_2 = \lambda_2$, with λ_1 and λ_2 real, so that we come to the following Hamiltonian

$$H_1 = \frac{\lambda_1}{2} \sum_{j=1}^{N-1} (c_j^\dagger c_{j+1} + c_j^\dagger c_{j+1}^\dagger + h.c.) \tag{1.108}$$

$$H_2 = \frac{\lambda_2}{2} \sum_{j=1}^{N-1} (c_j^\dagger c_{j+2} + c_j^\dagger c_{j+2}^\dagger + h.c.) \tag{1.109}$$

$$H_\mu = -\mu \sum_{j=1}^{N} (c_j^\dagger c_j - \frac{1}{2}) \tag{1.110}$$

$$H = H_\mu + H_1 + H_2 \tag{1.111}$$

Since all the parameters are real, the total Hamiltonian (1.111) still enjoys the same symmetries of the Kitaev chain with nearest neighbor coupling and thus belongs to the BDI topological class. In figure 1.2 it is shown the forming of Majorana edge modes for different hopping couplings for a more explicitly visual representation of the phenomenon.

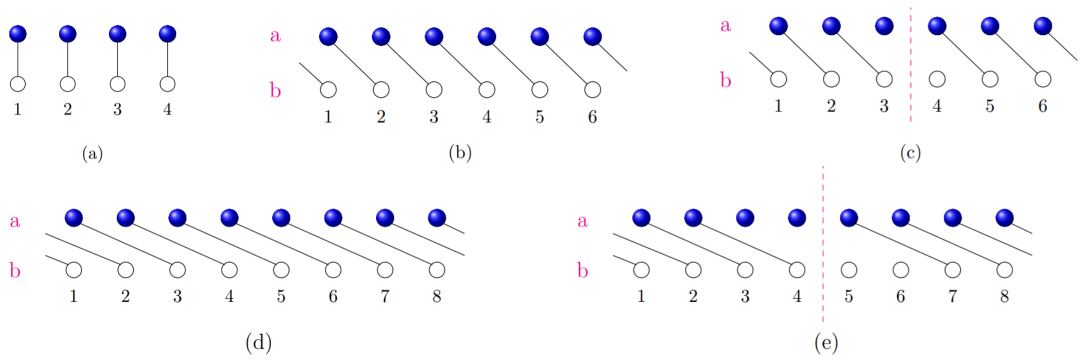


Figure 1.2: Different coupling realizations. The pictures show a fermionic chain where each site, numbered, is divided into two Majorana sites giving us two sublattices a and b. Panels (a), (b) and (d) show respectively no coupling, NN coupling and NNN coupling with the consequent formation of 0,1, and 2 edge modes. Panels (c) and (e) show a chain split in between separating into two open chains. Picture taken from [28].

We now switch to momentum space with a Fourier transformation assuming PBC are met

$$c_j = \frac{1}{\sqrt{N}} \sum_k e^{ikj} c_k \qquad\qquad c_j^\dagger = \frac{1}{\sqrt{N}} \sum_k e^{-ikj} c_k^\dagger \tag{1.112}$$

so that defining the new fermion operator $\Psi_k = (c_k, c_{-k}^\dagger)^T$ we can write

$$H = \frac{1}{2} \sum_k \Psi_k^\dagger \mathcal{H}_k \Psi_k$$

$$\mathcal{H}_k = [-\lambda_1 \sin(k) - \lambda_2 \sin(2k)]\sigma_y + [\lambda_1 \cos(k) + \lambda_2 \cos(2k) - \mu]\sigma_z \tag{1.113}$$

that is

$$\mathcal{H}_k = \begin{pmatrix} \lambda_1 \cos(k) + \lambda_2 \cos(2k) - \mu & -i\lambda_1 \sin(k) - i\lambda_2 \sin(2k) \\ i\lambda_1 \sin(k) + i\lambda_2 \sin(2k) & -\lambda_1 \cos(k) - \lambda_2 \cos(2k) + \mu \end{pmatrix} \tag{1.114}$$

where σ_α are Pauli matrices. This last equation can be put in the form $\boldsymbol{h}(k) \cdot \boldsymbol{\sigma}$ with

$$\boldsymbol{h}(k) = \begin{pmatrix} 0 \\ -\lambda_1 \sin(k) - \lambda_2 \sin(2k) \\ \lambda_1 \cos(k) + \lambda_2 \cos(2k) - \mu \end{pmatrix} \tag{1.115}$$

The windings around the origin of this vector define the topological phase the system belongs to.
Let us calculate the single particle energy spectrum. By diagonalizing (1.114) we obtain:

$$\epsilon(k) = \pm\sqrt{\mu^2 + \lambda_1^2 + \lambda_2^2 + 2\lambda_1(\mu - \lambda_2)\cos(k) - 2\lambda_2 \cos(2k)} \tag{1.116}$$

The energy gap closes for $\lambda_2 = \mu + \lambda_1$, $\lambda_2 = \mu - \lambda_1$ and for $\lambda_2 = -\mu$ for $|lambda_1| < 2|\mu|$. The positive sector of the phase diagram is showed in 1.3 with $\mu = 1$. To better understand the diagram 1.3, we first consider some limiting cases.

- $|\lambda_1|, |\lambda_2| \ll |\mu|$ is the trivial phase which hosts no Majorana edge modes

- $\lambda_2 = 0$ (the orizontal axis) we regain the Nearest Neighbor coupling Kitaev chain that we studied in the last section. Thus, we can see the phase transition from 0 edge modes to 1 occuring when λ_1, which is the hopping paramtere t, is larger then $\mu/2$.

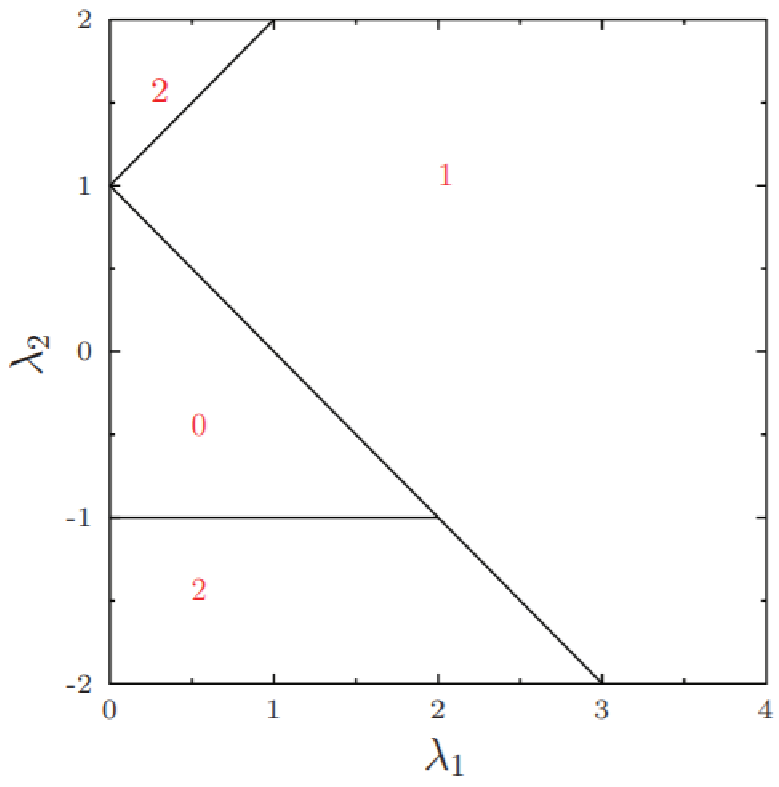


Figure 1.3: Phase diagram of the NNN Kitaev model. λ_1 and λ_2 are the parameters of the Hamiltonian (1.108). The red numbers represent different topological phases according to the count of Majorana edge modes. There are 0 edge modes for low values of the hopping paramterers as we can expect. One edge mode in the zone where the short range coupling is more relevant and two where the longer range term is more relevant and thus the chain is decomposed into two subchains, each one hosting one edge mode.

- $\lambda_1 = 0$ In this case there is only next to nearest neighbor coupling so the wire splits into two alternal Kitaev chains (the odd and even sites). Each chain has one Majorana edge mode appearing when $|\lambda_2| > \mu$.

We show now the appearence of the Majorana modes in the NNN Kitaev chain deriving their wavefunction.
Let us start by rewriting Hamiltonian (1.111) using the Majorana modes. For clarity we define $\gamma_{2j-1} = a_j$ and $\gamma_{2j} = b_j$, so that the Majorana Hamiltonian becomes

$$H = i \sum \left[\mu b_j a_j - \lambda_1 b_j a_{j+1} + \lambda_2 b_j a_{j+2}\right] \tag{1.117}$$

and we write it down in the basis $\psi^T = (a_1, b_1, a_2, b_2, \dots)$ using the Majorana anti-commutation relations $a_j, b_k = 0$, $a_j, a_k = \delta_{jk}$ for $j, k = 1, \dots, N$:

$$H = -\frac{i}{2} \begin{pmatrix} 0 & \mu & 0 & \dots & & & & \\ -\mu & 0 & \lambda_1 & 0 & \lambda_2 & \dots & & \\ 0 & -\lambda_1 & 0 & \mu & 0 & \dots & & \\ 0 & 0 & -\mu & 0 & \lambda_1 & 0 & \lambda_2 & \dots \\ \vdots & \vdots & & \ddots & \ddots & \ddots & & \end{pmatrix} \tag{1.118}$$

Correlation functions We now turn our attention to the calculation of correlation functions once again, since they will be largely exploited in the following chapters as our main data information.

Starting from the Fourier transformed Hamiltonian (1.4) and its one-particle matrix version (1.114) we want to put in a diagonal form with the eigenvalues (1.116) on its diagonal. In order to do that, we define new operators

$$\begin{pmatrix} \eta_k \\ \eta^\dagger_{-k} \end{pmatrix} \equiv U_k \begin{pmatrix} c_k \\ c^\dagger_{-k} \end{pmatrix} = \begin{pmatrix} \alpha_k & \beta_k \\ -\beta^*_k & \alpha^* \end{pmatrix} \begin{pmatrix} c_k \\ c^\dagger_{-k} \end{pmatrix} \tag{1.119}$$

for some unitary matrix U_k that we can put in a general form so that the new Hamiltonian is

$$H = \sum_k \begin{pmatrix} \eta^\dagger_k & \eta_{-k} \end{pmatrix} \begin{pmatrix} \varepsilon_k & 0 \\ 0 & -\varepsilon_k \end{pmatrix} \begin{pmatrix} \eta_k \\ \eta^\dagger_{-k} \end{pmatrix} \tag{1.120}$$

This requires that

$$U_k \mathcal{H}_k U^\dagger_k = \begin{pmatrix} \varepsilon_k & 0 \\ 0 & -\varepsilon_k \end{pmatrix} \tag{1.121}$$

In the one dimensional case we can write the unitary transformation setting $\alpha_k = \cos\theta_k$ and $\beta_k = i \sin\theta_k$ so that the last equation becomes

$$\begin{pmatrix} \cos\theta_k & i\sin\theta_k \\ i\sin\theta_k & \cos\theta_k \end{pmatrix} \begin{pmatrix} A & B \\ -B & -A \end{pmatrix} \begin{pmatrix} \cos\theta_k & -i\sin\theta_k \\ -i\sin\theta_k & \cos\theta_k \end{pmatrix} = \begin{pmatrix} \varepsilon_k & 0 \\ 0 & -\varepsilon_k \end{pmatrix} \tag{1.122}$$

Solving for the angle θ_k gives us the relation

$$\tan 2\theta_k = \frac{\lambda_1 \sin k + \lambda_2 \sin 2k}{\mu - \lambda_1 \cos k - \lambda_2 \cos 2k} \tag{1.123}$$

We can invert this relations to find θ_k and thus α_k and β_k. This completely determines the unitary matrix for the operators transformation.
In this way we introduced the Bogoliubov-de Gennes formalism in which the new operators, usually called Bogoliubov quasi-particles, are a combination of N annihilation operators of electrons, or creators of their "holes", and N creation operators of the same electrons. We thus doubled the degrees of freedom and that is the reason of having a double energy spectrum symmetric to the zero energy level.
In this new basis of quasi-particles the ground state of the system is given by the condition

$$\eta_k|GS\rangle = 0 \qquad\qquad k = \frac{2\pi l}{N}, l = 0, 1, \dots, N-1 \tag{1.124}$$

which can be shown to be a coherent state of superconducting Cooper pairs

$$|GS\rangle = \prod_k (\cos\theta_k + i\sin\theta_k c_k^\dagger c_{-k}^\dagger)|0\rangle \tag{1.125}$$

where $|0\rangle$ is the fermionic vacuum (*i.e.* $c_k|0\rangle = 0$ for all discretized momenta k).
Let us evaluate the correlation functions (1.94) on the Bogoliubov vacuum. Firstly, we go into momentum space through a Fourier transformation

$$C(j,l) = \langle c_j^\dagger c_l\rangle = \langle GS|c_j^\dagger c_l|GS\rangle = \frac{1}{N}\sum_k\sum_q e^{-ikj}e^{iql}\langle c_k^\dagger c_q\rangle \tag{1.126}$$

and then we invert equations (1.120) to express $c^\dagger$ and c in terms of $\eta^\dagger$ and η

$$= \frac{1}{N}\sum_k\sum_q e^{-ikj}e^{iql}\langle(-i\sin\theta_k\eta_{-k}^\dagger + \cos\theta_k\eta_k)(\cos\theta_q\eta_q - i\sin\theta_q\eta_{-q}^\dagger)\rangle \tag{1.127}$$

The only term that does not vanish is the expectation value $\langle\eta_k\eta_q^\dagger\rangle$, which leaves us with

$$\frac{1}{N}\sum_k e^{-ikj}e^{iql}\langle(-i\sin\theta_k\eta_{-k}^\dagger + \cos\theta_k\eta_k)(\cos\theta_q\eta_q - i\sin\theta_q\eta_{-q}^\dagger)\rangle \tag{1.128}$$

$$= \frac{1}{N}\sum_k e^{-ik(j-l)}(-\sin^2\theta_k) \tag{1.129}$$

$$= \frac{1}{N}\sum_k e^{-ik(j-l)}\left[\frac{\mu - \lambda_1\cos k - \lambda_2\cos 2k}{2\varepsilon(k)} - \frac{1}{2}\right] \tag{1.130}$$

The same calculation for the anomalous correlation function $F(j,l)$ gives us

$$F(j,l) = \langle c_j^\dagger c_l^\dagger \rangle = \frac{1}{N}\sum_k e^{-ik(j-l)}\left[-\frac{1}{2}\frac{\lambda_1 \sin k + \lambda_2 \sin 2k}{\varepsilon(k)}\right] \tag{1.131}$$

For both these correlation functions we are interested in their momentum space functions so we will assume from now on to work with $C(k)$ and $F(k)$, that is

$$\begin{aligned} C(k) &= \frac{\mu - \lambda_1 \cos k - \lambda_2 \cos 2k}{2\varepsilon(k)} - \frac{1}{2} \\ F(k) &= -\frac{1}{2}\frac{\lambda_1 \sin k + \lambda_2 \sin 2k}{\varepsilon(k)} \end{aligned} \tag{1.132}$$

1.5 The Kitaev Interacting Model

Let us consider a Kitaev Hamiltonian with an interacting term. It is a very interesting case of study due to the effects that the interaction has on the edge modes. Depending on the physical realization of the chain there are two main effects that can be detected. One effect is the suppression of the bulk gap by the interacting term, which makes the topological phase less stable. The other one is that the chemical-potential window over which Majorana modes exist is broaden by a repulsive potential [29].
The additional interacting term puts the Kitaev Hamiltonian in the form

$$H = \sum_i \left[-t\left(c_i c_{i+1}^\dagger + c_{i+1} c_i^\dagger\right) - \mu\left(c_i^\dagger c_i - \frac{1}{2}\right) + \Delta\left(c_i c_{i+1} + c_i^\dagger c_{i+1}^\dagger\right) + V n_i n_{i+1}\right] \tag{1.133}$$

where $n_i = c_i^\dagger c_i$ is the number operator for the fermionic site i, the interaction is repulsive for positive values of V and we obviously regain the original Kitaev Hamiltonian for $V = 0$. The introduction of the interacting term makes Hamiltonian (1.133) non bilinear in the creation/annihilation operators and that does not allow for any simple analytical solution. Therefore, numerical approaches to the problem are in order.
The interacting term does not allow for conservation of the totale fermion number, *i.e.* $[H, F] \neq 0$, where $F = \sum_i n_i$. However, the parity is still conserved.
Hamiltonian (1.133) is also symmetric under time reversal but not under charge-conjugation.

Phase Diagram The phase diagram of the interacting case can be obtained through numerical methods. We consider the work of R. Thomale et al. [30] who applied the Density Matrix Renormalization Group (DMRG) technique to calculate the single-particle density of state. In figure 1.4 the $V - \mu$ phase diagram is shown for $\Delta = 0.5$. The system can reside in 4 different phases separated by critical lines: a trivial and a topological superconducting phases (SC), and two Charge Density Wave phases (Incommensurate and Commensurate) which we will denote simply as Density Wave phases (DW).
The topological phase is characterized by the two-fold degeneracy of the ground state, it is the only phase with this property. The DW states have two groundstates in the same parity sector but their distinction is not relevant for our purposes. From now on they will both be considered as belonging to a trivial phase.
It is evident that for $V = 0$ we have the expected results for a non-interacting Kitaev chain, that is the gap closes at $\mu = 2$ breaking the groundstate degeneracy.

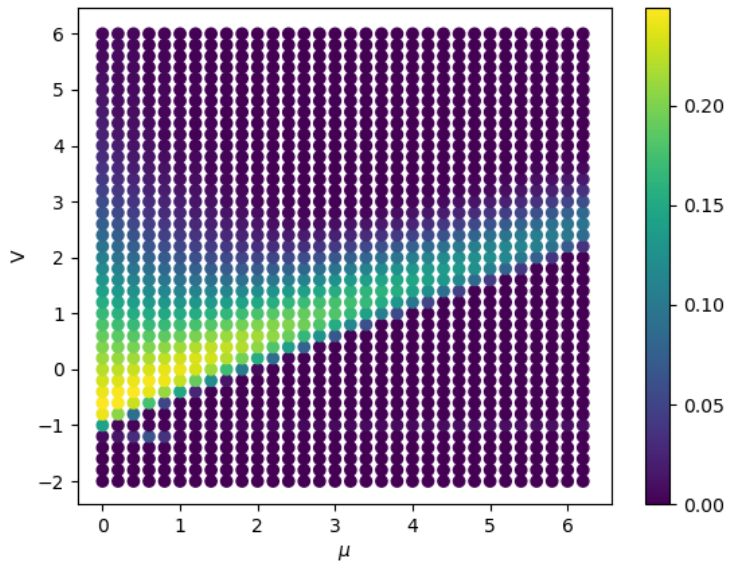


Figure 1.4: Phase diagram of the interacting Kitaev model obtained with the DMRG for $\Delta = 0$ and $t = 1$. Different colors represent topological phases. The yellow part with positive potential is the topological superconducting phase. This separates the two trivial sectors which are: the trivial superconducting phase at low potential and the Charge Density Wave (Incommensurate)

2 Machine Learning for Physics

Artificial Intelligence (AI) is a very commonly used word in these days and it actually describes an extremely large variety of fields of research that ranges from classical computation algorithms to cutting-edge technology [7].
While AI actually deals with any algorithm that, event faintly, resembles human cognitive behaviour, there is a subfield, called Machine Learning, which has gained a lot of attention from almost every scientific community in recent years.
Machine Learning (ML) and Deep Learning (DL) are hybrid fields of study ranging from statics to data science which aim at making predictions from data, trying to learn the rules that governs them [5].
Its extreme popularity in the last decades is certainly due to our recently achieved access to enormous amount of data, simply known as Big Data, which are necessary in order for ML and DL algorithms to work better then statistical methods [8].
Physics has always worked with exponentially increasing quantities of data, sometimes labelled as Extreme Data. It is, therefore, an extremely fertile field for these type of algorithms to be developed.
In addition to that, many of the basic concepts and techniques of ML - such as Monte-Carlo methods, simulated annealing, variational methods - have their origin in physics. Moreover, many deep learning algorithms are "energy-based" and they borrow many aspects from statistical physics [8].
Although these new techniques have had incredible results bringing us self-driving cars and almost-human image recognition, their success has not been followed by a complete scientific study which could validate it.
In this chapter we try to present the basic concepts of ML and DL that can be more interesting from a physicist's point of view. These ideas will be exploited in the last chapter to work on physical models trying to give a more scientific perspective

on the method used. We will mainly follow the review [8] which is addressed to physicists.

2.1 Key Concepts of Machine Learning

Machine Learning describes a vast field of algorithms and statistical methods that aim at making predictions learning from data.
Even though Machine and Deep Learning are two separated worlds they share many features and most of the results and techniques of ML also apply to DL. They are therefore very commonly confused, or intentionally put under the common name of Machine Learning for simplicity. The way any ML algorithm works is inherently different from any common computer algorithms. By that we mean that unlike "classical" computation problems which aim at providing results by having a dataset and a set of rules, ML tries to learn the rules knowing the data and their results [5].
While Machine Learning certainly stems from statistics, they clearly are two different brands of data science.
ML is the subfield of Artificial Intelligence which deals with predicting from data while statistics is more concerned with estimation of parameters from data. ML is also used when dealing with larger amount of data, compared to statistics.
The two fields still share a common framework. That is, working on an arbitrary quantity $\boldsymbol{x}$ we are interested in. This quantity depends on some parameters $\boldsymbol{\theta}$ of a model $p(\boldsymbol{x}|\boldsymbol{\theta})$ which describes the probability of observing $\boldsymbol{x}$ given $\boldsymbol{\theta}$. While statistics is more concerned in estimation of the parameters $\boldsymbol{\theta}$ which best fits the data, ML works on maximing the probability of predicting new data $p(\boldsymbol{x}|\boldsymbol{\theta})$.
Deep Learning is a subset of Machine Learning. We can state that one of the main differences between the two methods is in the knowledge of your data. ML tipically works with structured data, meaning tables of numerical values identified by labels - such as a list of information of clients of a specific service or the characteristics of houses locations and prices in a city. These data is very often preprocessed and *cleaned* in a way to make predictions easier. DL, instead, works with unstructured data that sometimes do not even go through preprocessing.
This distinction is not actually very strong. In general, the two methods share most of the common features. So, unless otherwise specified we - as much as the whole computer science community - will refer to Machine Learning as for both ML and DL.

2.1.1 Features of a Machine Learning algorithm

The first important thing to know before diving into the subject is that Machine Learning is traditionally split into three main subclasses according on how it works with the data. They are

- **Supervised Learning.** This class of algorithms works on known data, that is, the data is labelled either by a computer or a person.
  The scope of this methods is to understand the function that associate to the data their label. Classification problems are the perfect examples for this method. In physics a typical supervised problem would be to assign the correct phase to a system. Thus, the model can (has to) understand where the phase transition occurs.

- **Unsupervised Learning.** For this class of methods no *a priori* knowledge of the data is required. They are indeed suggested for use on unlabelled data which we do not know many information of. Algorithms of this class are commonly for clustering or dimensional reduction. The latter is particularly important in physics where problems often scale exponentially with the system size (the so called *curse of dimensionality*) [5].

- **Reinforcement Learning.** By this we mean all the algorithms that need to learn a specific set of rules. This models communicate with an external environment which gives them feedback on their behaviour. Starting from random sets of "moves", the algorithm updates its parameters so to learn the rules of the context. These types of algorithms are used for self-driving cars and robot science. In physics, the generative models exploit these techniques to compress the information of a wavefunction and describe the time evolution of a quantum many body state.

On top of these three methods we can have hybrid combinations such as *semi-supervised learning* and many others. Moreover, the line between the different *learnings* is usually pretty thin and no scientifically defined. So, the name we use is not as important as the previous knowledge we have of the data or the result of the algorithm.
The common three ingredients of almost every ML algorithm are

- **The dataset**. A set of independent and identically distributed data (so called *i.i.d hypothesis* [5]) and their labels. By *i.i.d* we assume the data to be generated by the same (often unknown) distribution. When dealing with a supervised learning problem we usuallly represent it by $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$ where $\boldsymbol{X}$ is a set of data of any type and $\boldsymbol{y}$ are the labels associated to it. For unsupervised learning, for example, there is no known label to assign to the data.

- **The model**. We can synthesize a ML algorithm as a function $f(\boldsymbol{X}; \theta)$ from $\boldsymbol{X}$ to $\boldsymbol{y}$ depending on some parameters θ which will be learned by the model to best fit the data.

- **The cost function**. A function $\mathcal{C}(\boldsymbol{y}, f(\boldsymbol{X}; \boldsymbol{\theta}))$ used to judge the precision of the algorithm depending on its task. An example is the Mean Squared Error (MSE) which is a simple average of the squared errors, that is $\mathcal{C}_{MSE} = \frac{1}{N}\sum_{i=1}^{N}(y_i - f(X_i; \boldsymbol{\theta}))^2$, where i runs over the test set elements.

The first measure is to split randomly the dataset into training data $\mathcal{D}_{\text{train}}$ and test data $\mathcal{D}_{\text{test}}$. The partition is usually $90\% - 10\%$. Fitting the model consists in finding the parameters $\hat{\boldsymbol{\theta}}$ which minimize the cost function only using the training set. The best set of parameters is then defined as $\hat{\boldsymbol{\theta}} = \arg\min_\theta \mathcal{C}(\boldsymbol{y}_{\text{train}}, f(\boldsymbol{X}_{\text{train}}; \boldsymbol{\theta}))$, where $\arg\min_\theta$ acts on a function (the loss function in our case) and returns the values $\boldsymbol{\theta}$ which minimizes it.
After the training, the model is then evaluated on the test set: $\mathcal{C}(\boldsymbol{y}_{\text{test}}, f(\boldsymbol{X}_{\text{test}}; \hat{\boldsymbol{\theta}}))$. This means that the same cost function that was minimized in the training process is evaluated on the test data (using the best parameters $\hat{\boldsymbol{\theta}}$ found during training) as a measure of how efficient the algorithm is at predicting new values. The latter is called the generalization or test error (or even cost or loss) $E_{out} = \mathcal{C}(\boldsymbol{y}_{\text{test}}, f(\boldsymbol{X}_{\text{test}}; \boldsymbol{\theta}))$, while the error on the training set is simply the value of the cost function $E_{in} = \mathcal{C}(\boldsymbol{y}_{\text{train}}, f(\boldsymbol{X}_{\text{train}}; \boldsymbol{\theta}))$. We can always expect $E_{out} \geq E_{in}$ because the model cannot fit new unseen data better then the ones it used for training. The test error E_{out} is the most common estimator for the performance of an algorithm. It is thus used to compare different ML methods.

2.1.2 Statistical Learning Theory methods

In the last section we saw how E_{in} and E_{out} give a numerical idea of the difference between fitting and predicting. We now want to give more insight into their relationship and how they are related to quantity such as the amount of data needed and the best model complexity.
The following concepts are borrowed from the field of Statistical Learning Theory

and Bayesian Inference. They are essential tools in giving a more precise meaning to the performance of ML algorithms, what it means to generalize to new data and what are the requirements in order to have the best prediction possible [5].
Let us start by considering a dataset $\mathcal{D} = (\boldsymbol{X}, \boldsymbol{y})$ and a function f such that $f(\boldsymbol{X}) = \boldsymbol{y}$. The task of ML is to find a function h such that $h \approx f$, if we do we can say we *learned* the model. By learned we mean that we expect the function h to work on test data approximately as well as it did on the training data [8].
If we suppose that our target function f cannot be learned exactly (which is often

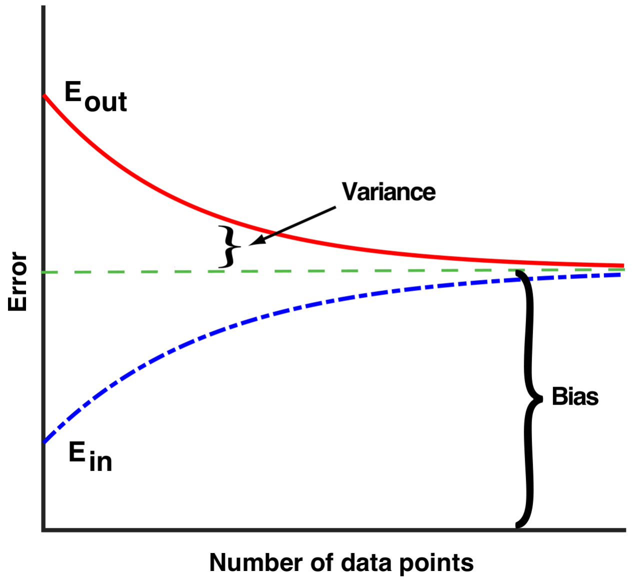
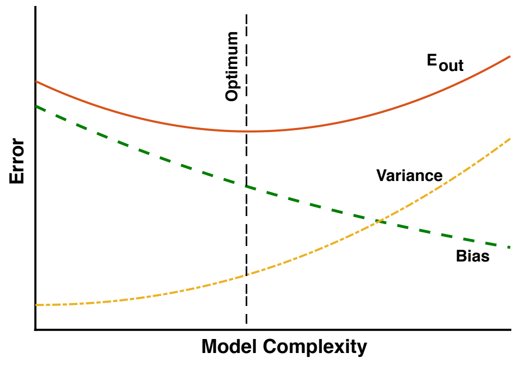
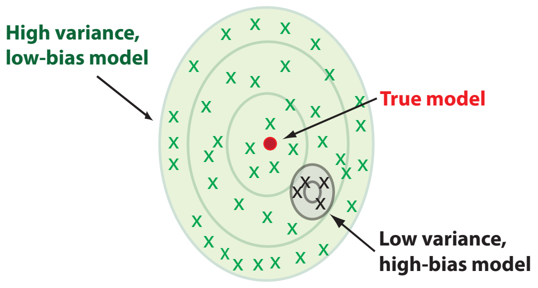


Figure 2.1: The picture shows the behaviour of the training and generalization error for a very large data set. It also shows the key elements to take into consideration for maximizing the efficiency of a model, that is the variance and the bias. Image taken from [8].

the case in particular in Deep algorithms) we can expect the training error E_{in} to become lower when increasing the data points. Then, since the function f cannot be learned, the E_{in} will start to increase with the amount of data. The test error instead will only decrease. Therefore, they will converge to the same error as the number of data points tends to infinity. A schematic of the process is presented in Figure 2.1.
We call the error in the infinite limit the bias. Which is the best our model can perform. The difference on how it performs on new data compared to the training set is the *variance* of the model. The variance is due to finite size effects of the dataset. That is, the noise which cannot be cancelled out with a limited amount of



(a)



(b)

Figure 2.2: The plot on the left side (a) shows how the test error, variance and bias change with model complexity. We can plot them together since they are all errors. Bias and variance grow inversely to each other, that is why there is need to find a balance. The figure on the right (b) exposes graphically the bias-variance tradeoff problem. A high-variance, low-bias model certainly recreates the output of the data model but with a high cost in precision, while the opposite might not exactly represent data but have good generalization error. Pictures taken from [8]

data.

The last quantity of interest in Figure 2.1 is the difference between test error and training error to be read as the mathematical difference between predicting and fitting data.

When the gap between E_{out} and E_{in} becomes very large we are dealing with the problem called *overfitting*. That means that our model learnt the features of the training set so well (low E_{in}) that it finds hard to recognize similar features on new data, even though the test set comes from the same distribution as the training set by hypothesis.

A similar issue, called *underfitting*, is simply the sign that the model is not learning well and it is thus given by large E_{in}. It is signaled by the non convergence of the loss function, meaning that E_in is too high.

Another statistical concept that is very important to us is deciding the complexity of a model in order to have the best performance. Complexity is sometimes defined qualitatively. In general, though, we can consider the number of parameters of an algorithm as its complexity.

Although improving the complexity of a model certainly diminishes its bias on a fixed dataset, it does not necessarily improve its performance. That is because on a

finite sized dataset an overly complex model would just enhance the variance.
Finding the best model is thus a balancing of increasing or reducing bias or variance. It is usually referenced as the *bias-variance tradeoff* and it is at the heart of why ML is difficult [8].
Figures 2.2a and 2.2b show qualitatively how to reach the optimal performance of an algorithm by complexity and what we mean by bias-variance tradeoff.

Bayesian Inference Bayesian Statistic is a field of studies that is born in the context of statistical learning but is set on very different premises. Statistics, in its frequentist approach, deals with making predictions based on an estimation of a parameter. Bayesian Inference, instead, takes into consideration all possible values of such (many) parameter(s) when making a prediction.
The Bayesian statistics uses probability to reflect degrees of certainty in states of knowledge [5]. This reasoning takes inspiration in statistical mechanics in which we interpret probability as descriptive of the behavior of large systems of which, in principle, the dynamics is not determined.
A Bayesian problem revolves around three distributions: the *prior distribution* $p(\boldsymbol{\theta})$ that collects any knowledge of the parameters that describe the data before we measure it, the *posterior distribution* $p(\boldsymbol{\theta}|\boldsymbol{X})$ which contains the information we can infer on the parameter after measuring the data and a likelihood function $p(\boldsymbol{X}|\boldsymbol{\theta})$ which is supposed to describe the data from the parameters $\boldsymbol{\theta}$. The three distributions are related by the famous Bayes rule:

$$p(\boldsymbol{\theta}|\boldsymbol{X}) = \frac{p(\boldsymbol{X}|\boldsymbol{\theta})p(\boldsymbol{\theta})}{\int d\boldsymbol{\theta}' p(\boldsymbol{X}|\boldsymbol{\theta}')p(\boldsymbol{\theta}'))} \tag{2.1}$$

The denominator is the distribution of the data, an often intractable function which needs to be simulated by Markov Chain Monte Carlo methods. The bayesian inference is strongly connected to Machine Learning because its nature is to integrate data with domain knowledge of the model environment [31].
A common feature of statistical and Bayes inference is the *Maximum Likelihood Estimation* (MLE). In this view, one selects the parameters that maximize the log probability of observing the data once the parameters are known, that is:

$$\hat{\boldsymbol{\theta}} = \arg\max_{\boldsymbol{\theta}} \log p(\boldsymbol{X}|\boldsymbol{\theta}). \tag{2.2}$$

This equation is at the heart of Machine Learning. In fact, we will see soon that the cost function is interpreted as the negative log probability of observing the data. Minimizing that quantity is the same as the maximation of equation (2.2).
While this approach is common in the field of statistics, the knowledge of the prior distribution is a unique bayesian feature. The priors are usually defined as either *informative* or *uninformative* according to the degree of information they provide on the parameters $\boldsymbol{\theta}$.
In ML we usually deal with informative priors and we can give a quantitative measure to this knowledge. That is done by substituting $p(\theta)$ (our knowledge of the parameters) in equation (2.1) with the prior $p(\theta|\lambda)$ which is a distribution of the parameters that we can set based on what we expect. For example if we expect many parameters to be around zero - which is sensible for models with tens of thousands of parameters like ours - we will have a Gaussian prior $p(\boldsymbol{\theta}|\lambda) = \prod_j \sqrt{\frac{\lambda}{2\pi}} e^{-\lambda\theta_j^2}$. If we expect most of them to be zero a typical prior is the Poisson distribution, *i.e.* $p(\boldsymbol{\theta}|\lambda) = \prod_j \frac{\lambda}{2} e^{-\lambda|\theta_j|}$. We will see soon the meaning of the parameter λ in an example.
Up to now we gave a meaning to the calculation of the probability $p(\boldsymbol{\theta}|\boldsymbol{X})$, that in ML we see as the estimation of the best parameters to fit our model. Yet, we need to actually calculate the set of parameters $\boldsymbol{\theta}$ from our estimation. This is commonly done [30] by calculating the *maximum-a-posteriori* estimate

$$\hat{\boldsymbol{\theta}}_{MAP} = \arg\max_{\boldsymbol{\theta}} p(\boldsymbol{\theta}|\boldsymbol{X}). \tag{2.3}$$

because it is more simple then actually evaluating the Bayes equation (2.1).
We can apply MLE and MAP to a linear regression problem to show their meaning. Let us consider a set of points generated by some function $\boldsymbol{y} = \boldsymbol{x^T w} + \epsilon$, where ϵ is gaussian noise with mean 0 and variance σ^2, $\boldsymbol{w}$ parameters which need to be learned and all together they are the parameters $\boldsymbol{\theta}$ of the model. To implement a bayesian reasoning we write

$$p(y|\boldsymbol{x}, \boldsymbol{\theta}) = \mathcal{N}(y|\mu(\boldsymbol{x}), \sigma(\boldsymbol{x})) \tag{2.4}$$

where $\mu(\boldsymbol{x}) = \boldsymbol{x^T w}$, $\sigma(\boldsymbol{x}) = \sigma^2$ and $\mathcal{N}$ stands for Normal Distribution. Now, applying MLE we calculate

$$\hat{\boldsymbol{\theta}} = \arg\max_{\theta} \log p(\mathcal{D}|\boldsymbol{\theta}) = \arg\max_{\theta} l(\boldsymbol{\theta}) \tag{2.5}$$

where, assuming the *i.i.d* hypothesis are true

$$l(\boldsymbol{\theta}) = \sum_{i=1}^{n} \log p(y_i|\boldsymbol{x}^{(i)}, \boldsymbol{\theta}) = -\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)^2 - \frac{n}{2}\log\left(2\pi\sigma^2\right) \tag{2.6}$$

As a result, minimizing the log likelihood of the data distribution is the same as performing least squares estimation (except for a constant value which we will discard).
We already mentioned that instead of maximizing the likelihood we just calculated it is easier to calculate the MAP estimator. Recalling Bayes equation

$$p(\boldsymbol{\theta}|\mathcal{D}) \propto p(\mathcal{D}|\boldsymbol{\theta})p(\boldsymbol{\theta}) \tag{2.7}$$

the MAP calculation becomes

$$\hat{\boldsymbol{\theta}}_{MAP} \equiv \arg\max_{\theta} \log p(\boldsymbol{\theta}|\mathcal{D}) = \arg\max_{\theta} \log p(\mathcal{D}|\boldsymbol{\theta}) + \log p(\boldsymbol{\theta}). \tag{2.8}$$

Considering a Gaussian prior with variance τ^2 we get

$$\hat{\boldsymbol{\theta}}_{MAP} \equiv \arg\max_{\boldsymbol{\theta}} \left[-\frac{1}{2\sigma^2}\sum_{i=1}^{n}\left(y_i - \boldsymbol{w}^T\boldsymbol{x}^{(i)}\right)^2 - \frac{1}{2\tau^2}\sum_{j=1}^{n} w_j^2 \right]. \tag{2.9}$$

This calculation defines the best cost function one can devise for a regression problem. It consists in maximizing the negative sum of the square losses (individual differences between the prediction of the model y_i on a single data point $\boldsymbol{x}^{(i)}$) and the negative sum of the squared weights so that both quantities tend to zero.
The second term is a *regularization* term called L_2 regularization, because it controls the square module of the weights in order to minimize them. Had we used a Poisson prior, we would have gotten the L_1 regularization, *i.e.* $\lambda\sum_i |w_i|$. In both cases a new parameter is added to the model and needs to be set. They are the first two example of *hyperparameters* of a ML algorithm. We will see in the next section of to find their best values, though they will not be relevant in our final results.

2.1.3 Gradient Descent Algorithms

We already mentioned that every ML problem is based on minimizing a cost function by adjusting its parameters according to data.
The minimization of the cost function is based on changing the parameters in the

direction where the gradient of the cost function is larger and negative.
The objective is obviously to reach a minimum for the cost function often called error function since it is a global function which needs to be minimized. Unfortunately, this target might be very complicated to achieve because in ML and DL we work with high-dimensional intricate and usually unknown functions that present many local minima.
Let us consider an energy function $E(\boldsymbol{\theta})$, with $\boldsymbol{\theta}$ its parameters, written as a sum of terms over n elements of a dataset

$$E(\boldsymbol{\theta}) = \sum_{i=1}^{n} e_i(\boldsymbol{x}_i, \boldsymbol{\theta}), \tag{2.10}$$

where e_i could be a squared difference or log probability *etc.* Mathematically speaking the iterative process for GD at time t is

$$\boldsymbol{v}_t = \eta_t \nabla_\theta E(\boldsymbol{\theta}_t), \tag{2.11}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{v}_t \tag{2.12}$$

where $\boldsymbol{v}$ is the directional derivative weighted by the *learning rate* η_t, an hyperparameter which determines how much the parameters should change along the gradient direction.
If the learning rate is sufficiently small, the optimization procedure should take the error function to its local minimum [8] but that comes with a high computational cost because more steps would be needed for convergence. On the other hand, a too large η might produce the opposite effect making the energy function skip the minimum altogether. One can show that in the case of a quadratic energy function, the optimal learning rate to reach the minimum is given by the second derivative of the cost function [32]. This is indeed in agreement with the traditional Newton's method for finding the minimum of a function.
The generalization of this approach to multi-dimensional functions involves calculating the Hessian matrix. This is computationally too expensive for algorithms that work with large datasets such as ours, it is therefore important to find first order approximation to the GD problem.
Although finding the right learning rate is one of the main problems when dealing with a ML algorithm, there are many other issues related to GD that we can summarize in this way:

- **Local Minima**. Since GD is deterministic it might always converge to the same local minima and never be able to increase its performance. This problem, in physics, is avoided by simulated annealing with the introduction of stochasticity which is what we need for GD.

- **Computational Cost**. Since most energy functions depend on values summed over all the data points it is necessary to find a different way to calculate the gradient.

- **Sensitivity**. The GD can vary significantly according to the choice of learning rate and initial conditions.

The solution to this problems is called Stochastic Gradient Descent (SGD). Under this name rely most of the ML algorithms used at present. Stochasticity is introduced calculating the gradient of the cost function over a small portion of the dataset, a *mini-batch*. Mini-batches are usually much smaller then the original dataset. Let us divide the n points of the dataset in M mini-batches so that now the gradient of the energy function (2.10) becomes

$$E^{BM}(\boldsymbol{\theta}) = \sum_{i \in B_k} \nabla_\theta e_i(\boldsymbol{x}_i, \boldsymbol{\theta}), \tag{2.13}$$

where B_k is the $k-th$ minibatch and $k = 1, 2, \dots, n/M$. With this approach we still follow the steps (2.11) but using E^{BM} instead. The benefits of this algorithm are multiple: it introduces a non deterministic dynamic so that the algorithm does not get stuck in local minima, it speeds up calculations and it is also thought to act a natural regularizer against overfitting in deep structured networks [33]. In practice SGD averages over the data points of a minibatch. It is important that the training data is shuffled before applying this procedure. In this way, at each iteration, the upgrade of the model's parameters is calculated taking into account data with different labels and thus the step is more likely directed in the direction of a global minima. If we did not shuffle the data there might be a mini batch made up only of similar data (with same label) and the next step would be biased because it would not average the other types of data (with a different label). So, we repeat that in order for this approach to work it is necessary that the dataset is randomized so that each minibatch contains as many different points as possible.
With gradient descent we can explain how the training of most ML algorithms work. Once the model is decided and the dataset split into training and testing, the

parameters are initialized randomly. Then for every batch of the training set the upgrades are calculated and the parameters updated. Clearly the dimension of the batch size determines how often this happens. Once the SGD has swept the whole training data we say that one *epoch* of training is completed. It is customary to let the SGD train for many epochs.
(Mini)Batch size and number of epochs are two hyperparameters, together with the L_1, L_2 regularizers we saw in the last section and the learning rate η. In order to determine the best values for the hyperparameters it is customary to split the training data once again in a training and *validation* part. The training data is still used for gradient descent, the validation data is instead used as a intermediate test set, to evaluate the efficiency of the algorithm on new unseen data after every epoch. In this way we have to check if both the cost function calculated on the training and the validation set goes to zero with increasing number of epochs. When neither of them approaches zero we fall into underfitting. When training loss goes to zero but the validation does not we encounter overfitting. That is, the algorithm learns the training data so well that it cannot generalize to new one. We will see in Chapter 3 that controlling the number of epochs and the batch size is particularly important.

Adding momentum Another important feature we can add to SGD is momentum. Quantitatively it consists in adding a memory term γ, $0 \leq \gamma \leq 1$ that keeps track of the behavior of the algorithm, namely

$$\boldsymbol{v}_t = \gamma \boldsymbol{v}_{t-1} + \eta_t \nabla_\theta E(\boldsymbol{\theta}_t), \tag{2.14}$$

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \boldsymbol{v}_t. \tag{2.15}$$

An equivalent discretized version of this is

$$\Delta\boldsymbol{\theta}_{t+1} = \gamma \Delta\boldsymbol{\theta}_t - \eta_t \nabla_\theta E(\boldsymbol{\theta}_t), \tag{2.16}$$

where $\Delta\boldsymbol{\theta} = \boldsymbol{\theta}_t - \boldsymbol{\theta}_{t-1}$. This formula is useful to make a comparison with a simple physical model of a particle with mass m moving in a viscous medium with damping coefficient μ and potential $E(\boldsymbol{w})$, with $\boldsymbol{w}$ the particle's position [34].
The equation of motion of the particle is

$$m\frac{d^2\boldsymbol{w}}{dt^2} + \mu\frac{d\boldsymbol{w}}{dt} = -\nabla_w E(\boldsymbol{w}), \tag{2.17}$$

and it can be cast into a discretized form [1] as

$$m\frac{\boldsymbol{w}_{t+\Delta t} - 2\boldsymbol{w}_t + \boldsymbol{w}_{t-\Delta t}}{(\Delta t)^2} + \mu\frac{\boldsymbol{w}_{t+\Delta t} - \boldsymbol{w}_t}{\Delta t} = -\nabla_w E(\boldsymbol{w}). \tag{2.18}$$

Inverting the equation we can rewrite it like

$$\Delta\boldsymbol{w}_{t+\Delta t} = -\frac{(\Delta t)^2}{m+\mu\Delta t}\nabla_w E(\boldsymbol{w}) + \frac{m}{m+\mu\Delta t}\Delta\boldsymbol{w}_t. \tag{2.19}$$

If we consider the position $\boldsymbol{w}$ as the parameters θ we can compare it to (2.16) so to identify

$$\gamma = \frac{m}{m+\mu\Delta t}, \qquad \eta = \frac{(\Delta t)^2}{m+\mu\Delta t}. \tag{2.20}$$

In this picture the momentum γ has the same behavior of a mass, *i.e.* providing inertia, and both γ and the learning rate η depend on the viscous parameter.
The meaning of this is that adding momentum allows the SGD algorithm to move faster in momentum space along directions with persistent yet small gradients. This is useful in the initial phase of training an algorithm where the landscape of the model function can be very flat. In fact momentum increases the speed along flat parts and damps down oscillations on steeper paths with higher curvature.
Most of modern SDG algorithms exploit all the features we just introduced and often also rely on momentum that multiply the square of the gradient of the cost function. A recently introduced and extremely popular one is the ADAM algorithm for SGD [35]. This algorithm, which we employed for our results in Chapter 3, keeps a running average of the gradient. So that the gradient has larger updates in the directions were the cost function changes more. We refer the reader to a more detailed explanation in [35].

2.1.4 Regression for Classification Problems

Some of the most popular and successful algorithms in Learning Theory resolve classification problems. This is particularly true in Deep Learning data science environments, for example in image recognition, but also in physical context. A typical problem in this category is classification of phases of matter. It can be used in simple

[1] we can define a *central difference* as $\delta_h[f](x) = f(x+\frac{1}{2}h) - f(x-\frac{1}{2}h)$ so that the central second derivative is simply $f''(x) \approx \frac{\delta_h^2[f](x)}{h^2} = \left(\frac{f(x+h)-f(x)}{h} - \frac{f(x)-f(x+h)}{h}\right)/h = \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$

cases such as the Ising 2D classical model [12] (as we will see in section 2.3) or more complicated topological phases classifications which need the use of Deep architectures [14]. Classification algorithms are especially important to us since the aim of this thesis is classifying topological phases of matter. That is why it is important to spend some words on how and why a ML or DL classifier works.
The most simple classifier is called *perceptron*. It is the the parent of the present *neuron*, the core structure of every deep learning algorithm which we will see in the next section.
An implementation of a perceptron for binary classification performs a linear transformation of the input data $\boldsymbol{x}_i$

$$s_i = \boldsymbol{x}_i^T \boldsymbol{w} + b_0 \tag{2.21}$$

and passes it through a sign function in order to get a 0 or a 1 in output.
In most cases, though, we are interested in an output that states the probability that a certain data point belongs to a class. In order to obtain a probability rather than a "hard" classification we pass the linear combination through an *activation function* which is a nonlinear function whose outputs can be seen as probabilities. Although there are several activation functions available, which we will present in the next section, we now focus on two cases: the logistic function or sigmoid and the SoftMax function.

Logistic Function for Binary Classification The sigmoid function is an activation of the input s such that

$$\sigma(s) = \frac{1}{1+e^{-s}}. \tag{2.22}$$

By using the function (2.22) an algorithm can output the probability for a datapoint $\boldsymbol{x}_i$ to belong to one of two classes $y_i = 0, 1$ as

$$\begin{aligned} P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\theta}) &= \frac{1}{1+e^{-\boldsymbol{x}_i^T\boldsymbol{\theta}}}, \\ P(y_i = 0|\boldsymbol{x}_i, \boldsymbol{\theta}) &= 1 - P(y_i = 1|\boldsymbol{x}_i, \boldsymbol{\theta}), \end{aligned} \tag{2.23}$$

where $\boldsymbol{\theta} = \boldsymbol{w}$ is the set of parameters that need to be learned, we will assume to work only on weights and no bias term for simplicity.
Notice that these probabilities are the same for a two-state system of being in either

one of the two states if their energy difference were $\Delta\epsilon = \boldsymbol{x}_i^T\boldsymbol{\theta}$ [8].
To determine the cost function of a classifier that uses the sigmoid activation we use the Maximum Likelihood Estimation (MLE) as seen in section 2.1.2. The probability of observing a set of N data $\mathcal{D}$ is

$$P(\mathcal{D}|\boldsymbol{w}) = \prod_{i=1}^{N} [\sigma(\boldsymbol{x}_i\boldsymbol{w})]^{y_i}[1-\sigma(\boldsymbol{x}_i\boldsymbol{w})]^{1-y_i}. \tag{2.24}$$

Where from now on we assume that the product $\boldsymbol{x}\boldsymbol{w}$ is element-wise. Then its log-likelihood is

$$l(\boldsymbol{w}) = \sum_{i=1}^{N} y_i \log\sigma(\boldsymbol{x}_i^T\boldsymbol{w}) + (1-y_i)\log[1-\sigma(\boldsymbol{x}_i^T\boldsymbol{w})] \tag{2.25}$$

and the maximum likelihood estimator (2.2) is the set of parameters that maximizes (2.25), that is:

$$\hat{\boldsymbol{w}} = \arg\max_{\boldsymbol{\theta}} \sum_{i=1}^{N} y_i \log\sigma(\boldsymbol{x}_i^T\boldsymbol{w}) + (1-y_i)\log[1-\sigma(\boldsymbol{x}_i^T\boldsymbol{w})] \tag{2.26}$$

while the cost, or error, function its simply the negative log-likelihood, that is

$$\mathcal{C}(\boldsymbol{w}) = -l(\boldsymbol{w}) = \sum_{i=1}^{N} -y_i \log\sigma(\boldsymbol{x}_i^T\boldsymbol{w}) - (1-y_i)\log[1-\sigma(\boldsymbol{x}_i^T\boldsymbol{w})] \tag{2.27}$$

the r.h.s. of equation (2.27) is known as the *cross-entropy.*
To minimize this function we take its gradient with respect to the weights $\boldsymbol{w}$. Let

us call $z = \boldsymbol{x}_i^T \boldsymbol{w}$ and using the relation $\partial_{\boldsymbol{w}} \sigma(z) = \sigma(z)[1-\sigma(z)]\boldsymbol{x}_i$ we obtain

$$\begin{aligned}
\nabla_{\boldsymbol{w}} \mathcal{C} &= \sum_{i=1}^{N} -y_i \nabla_{\boldsymbol{w}} \log \sigma(z) - (1-y_i)\nabla_{\boldsymbol{w}} \log[1-\sigma(z)] \\
&= \sum_{i=1}^{N} -y_i \frac{1}{\sigma(z)} \sigma(z)[1-\sigma(z)]\boldsymbol{x}_i - (1-y_i)\frac{1}{1-\sigma(z)}[-\sigma(z)(1-\sigma(z))]\boldsymbol{x}_i \\
&= \sum_{i=1}^{N} -y_i[1-\sigma(z)]\boldsymbol{x}_i - y_i \sigma(z)\boldsymbol{x}_i + \sigma(z)\boldsymbol{x}_i \\
&= \sum_{i=1}^{N} [\sigma(z) - y_i]\boldsymbol{x}_i.
\end{aligned} \tag{2.28}$$

This relation cannot be written in a closed form [8] so it needs numerical methods for the gradient descent such as the ones we presented in the last section. An example of binary classification using a sigmoid will be presented in section 2.3.

SoftMax function for multi-class classification. In the case, such as this thesis, where there is need to divide a dataset into multiple classes we can generalize logistic regression. Let us call y_i the label of a datapoint $\boldsymbol{x}_i$ which can assume M values, $y_i \in 0, 1, \ldots, M-1$, we define a vector with all zeroes except for the value category m of y_i, namely

$$y_{im} = \begin{cases} 1, & \text{if} \quad y_i = m \\ 0, & otherwise \end{cases} \tag{2.29}$$

Thus, the probability for $\boldsymbol{x}_i$ to belong to the class m' is

$$P(y_{im'} | \boldsymbol{x}_i, \{\boldsymbol{w_k}\}_{k=0}^{M-1}) = \frac{e^{\boldsymbol{x}_i^T \boldsymbol{w}_{m'}}}{\sum_{m=0}^{M-1} e^{\boldsymbol{x}_i^T \boldsymbol{w}_m}} \tag{2.30}$$

which is known as the SoftMax activation function. This function accepts in input a vector of values and simply makes the largest value much larger then the rest of the input points, which contribution is damped exponentially.
The likelihood of this classifier is

$$P(\mathcal{D}|\boldsymbol{w}_k{}_{k=0}^{M-1}) = \prod_{i=1}^{N} \prod_{m=0}^{M-1} [P(y_{im}=1|\boldsymbol{x}_i, \boldsymbol{w}_m)]^{y_{im}} [1 - P(y_{im}=1|\boldsymbol{x}_i, \boldsymbol{w}_m)]^{1-y_{im}}, \tag{2.31}$$

so its negative logarithm is the cost function to minimize, that is:

$$\mathcal{C}(\boldsymbol{w}) = -\sum_{i=1}^{N}\sum_{m=0}^{M-1} y_{im}\log P(y_{im}=1|\boldsymbol{x}_i,\boldsymbol{w}_m) + (1-y_{im})P(y_{im}=1|\boldsymbol{x}_i,\boldsymbol{w}_m) \quad (2.32)$$

which is identical to (2.27) for M = 1 but whose gradient is more complicated.

2.2 Deep Learning

The popularity of Machine Learning is given by its simplicity and its success on a vast variety of tasks. Yet, there is a set of problems such as speech recognition or image classification, in which ML finds many difficulties. It is the search of more powerful algorithms that led the AI world to the development and study of Deep Learning [5].
The underlying architecture of any DL algorithm is the so called (Deep) Neural Newtork (DNN). These type of structures have roots in the last century but became popular only in the last decade thanks to a series of impressive results on tasks where ML would greatly fail [8].
The fast development of the implementation of DNN was not though followed by a complementary study of the theory behind them. That is why it is often unclear how a DL algorithm can perform a task or what is exactly "learns". This problem is often referred to as the computational *black box* that is every neural network.
Deep Learning is massively used nowadays in many fields of study, including mathematics and physics. The effort of these communities is not only directed to solving new problems through the help of DL but also to try to understand why it works.

2.2.1 Deep Neural Networks (DNN)

A Neural Network is a deep learning structure that acts on an input with a series of non-linear transformations. It is meant to be trained in a supervised environment and mostly used for classification problems. The building block of these type of algorithms is the neuron. As we already mentioned, the first example of a neuron was proposed by Rosenblatt [36] under the name of *perceptron*. His idea was to reproduce the action of a human neuron which takes acts on a linear combination of some input data and outputs a value after acting with a non-linear function on it. A graphical representation is shown in Fig 2.3.

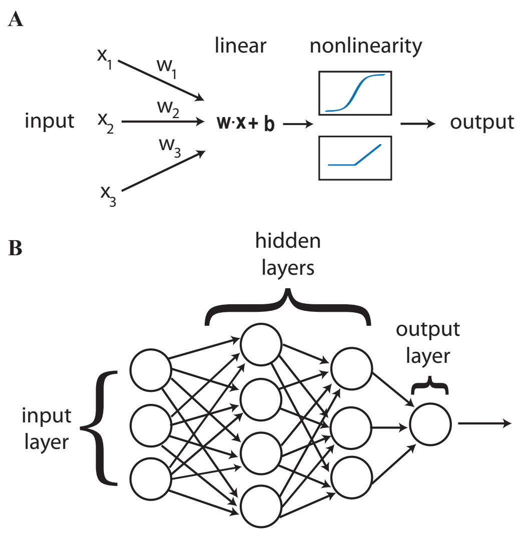


Figure 2.3: Representation of a single neuron (A) and the structure of a Neural Network (B). The neuron has a set of weights and a bias value that transform the input and pass it through some non-linear function, see below, to produce an output. The structure of a Deep Neural Network is made out of stacked neurons, forming a layer, and several layers following each other accepting the output of the previous layer as their own input. Picture taken from [8]

.

If we stack more neurons working on the same data but taking in a different linear combination of them we produce a layer. Stacking more layers one after the other produces the "deep" architecture of a Deep Neural Network which is shown in a simplified version in Figure 2.3. These layers are called *hidden layers* and they all transform the input in a non-linear fashion until the last layer which produces the output and it is known as the output layer. Each layer is connected through the previous layer by a weight matrix and a bias vector which take the output of a layer and transform it into the input of the following layer. For this reason a DNN can be seen as a complicated nonlinear transformation of the input into an output that depends on the parameters of every layer [8]. It is also the reason why this network are called Fully Connected. The introduction of non-linearity in the network is extremely important because it makes it highly non-trivial, non-deterministic and more thus powerful in the task of generalization which is the purpose of every good DL algorithm. If we consider the network as a mapping ϕ of the input into the output, we can think og ϕ as providing a set of features to describe the input [5]. This gives a first intuition about the differences between ML and DL. The first one works on inputs to directly produce an output while the latter learns characteristics

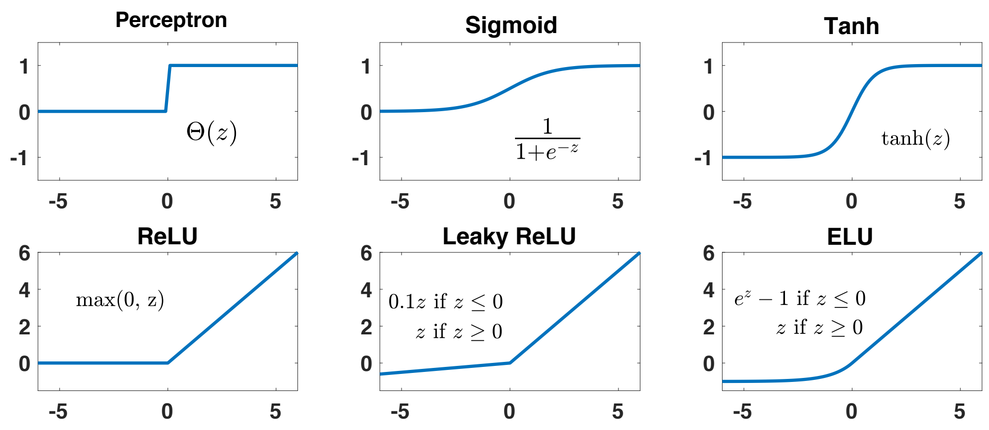


Figure 2.4: Different activation functions. The first row shows activations that are becoming obsolete and substituted with the second row. The Perceptron is a step function, often known as a "hard" classifier. The Sigmoid gives less importance to negative values, producing a zero. Tanh gives equal but opposite value to positive and negative inputs. These last three clearly saturate pretty quickly giving the same importance to very large or very small values. The second row shows more recent activations, widely used for their simplicity in the optimazation of the DNN's, see Chapter 3. Picture taken from [8]

.

of the input and works on them to produce the output. This subtle but important difference is the key to the generalization power of DL algorithms.
As already mentioned, nonlinearity is introduced through activations functions. Traditional ones are the sigmoid or logistic function or the *tanh* function. The first one produce an output between 0 and 1 while the latter between -1 and 1. In recent years the AI community started using more simple non-linear structures such a ReLU activation function which gives out 0 for every negative input. A comparison is shown in Fig 2.4.
The power of neural networks is encoded in the universal approximation theorem which states that a neural network can approximate any function with the cost of an at most exponential growth in the number of neurons [5]. Having said that, the complexity of a neural network does not always come with a greater performance for all the reasons mentioned in section 2.1.2. Although we can aspect a model to reduce its bias increasing its complexity it is commonly found that selecting the right dimension of NN is based on intuition and is often problem-specific [8].

Training a DNN The procedure to train a neural network follows the same basic step of every Machine Learning algorithm: define a cost/loss function, minimize it through gradient descent (or possibly, any other technique), update the parameters (weights) of the algorithm in order for the cost function to decrease. Most common cost functions for training DNN are the Mean Squared Error

$$E(\boldsymbol{w}) = \frac{1}{N}\sum_{i=1}^{N}(y_i - \hat{y}_i(\boldsymbol{w}))^2, \tag{2.33}$$

or the binary/categorical cross entropy discussed in the last section.
In the training process, it is required to calculate the derivative of the cost function w.r.t. every parameter of a model at each training step. This is not a feasable task for neural networks because common DNN can have tens of thousands up to millions of parameters. This issue firstly held back the study of this algorithms but was resolved with the introduction of backpropagation [37].
Backpropagation, or backprop, is nothing but a simple application of the chain rule of calculus. It is important, though, to have a mathematical insight of what happens when a DNN is trained and what or why it actually learns something.
We will use a standard notation for the neural network architectures. Let us assume there are L layers in our network and define the weights w^l_{jk} and the biases b^l_j as the weights and biases connecting the k-th neuron of the $l-1$-th layer to the j-th neuron of the l-th layer. We call the activations a^l as the output after the nonlinear function σ of the layer l. Thus, they are related to the previous layer by

$$a^l_j = \sigma\big(\sum_k w^l_{jk}a^{l-1}_k + b^l_j\big) = \sigma(z^l_j) \tag{2.34}$$

where we defined the affine transformation

$$z^l_j = \sum_k w^l_{jk}a^{l-1}_k + b^l_j \tag{2.35}$$

The cost function is a function of the last layer activation such as $\mathcal{C} = \sum_{j=0}^{N_L-1}\text{cost}(a^L, \hat{y})$. Let us define the error associated to the j-th neuron of layer l as the partial derivative of the cost function w.r.t. the weighted input z^l_j as

$$\Delta^l_j = \frac{\partial\mathcal{C}}{\partial z^l_j} = \frac{\partial\mathcal{C}}{\partial a^l_j}\sigma'(z^l_j) \tag{2.36}$$

We notice that this is also the partial derivative of the cost function w.r.t. the bias b^l_j since $\partial b^l_j / \partial z^l_j = 1$. Using the chain rule we can write

$$\Delta^l_j = \frac{\partial \mathcal{C}}{\partial z^l_j} = \sum_k \frac{\partial \mathcal{C}}{\partial z^{l+1}_j} \frac{\partial z^{l+1}_j}{\partial z^l_j} = \sum_k \Delta^{l+1}_k \frac{\partial z^{l+1}_j}{\partial z^l_j} = \left(\sum_k^{l+1} w^{l+1}_{kj} \right) \sigma'(z^l_j) \tag{2.37}$$

Finally, the partial derivative of the cost function with respect to the network weights is

$$\frac{\partial \mathcal{C}}{\partial w^l_{jk}} = \frac{\partial \mathcal{C}}{\partial z^l_j} \frac{\partial z^l_j}{\partial w^l_{jk}} \tag{2.38}$$

Equations (2.36), (2.37) and (2.38) can be used to calculate any partial derivatives with respect to the parameters of the network, and thus the gradient of the cost function.
A sketch of the workflow of the algorithm is **The Backpropagation Algorithm**

1. **Activation at input layer.** Calculate the activations a^1_j of the first layer,

2. **FeedForward.** Produce the activations of all the layers until the output layer, this is also called the "forward pass"

3. **Error at top layer** Calculate the error of the top layer using equation (2.36),

4. **Backpropagate the error.** Use equation (2.38) to propagate the error backwards and calculate Δ^l_j for all layers

5. **Calculate gradient.** Use equations (2.36) and (2.38) to calculate the partial derivatives of the cost function w.r.t. the weights and the biases of all the layers

We can see how the algorithm is called backpropagation, it relies on calculating all the activations of a network and then going backwards to calculate all the contributions to the partial derivatives of the total gradient. The fact that the information is passed forward gives these types of networks the name FeedForward Neural Networks.

2.2.2 Convolutional Neural Newtorks (CNN)

A common feature in all the fields of study of physics is the exploitation of symmetries and invariances of the system. Very often even the dataset that we feed

to neural networks have local features or global characteristics which are invariant to translations, rotations or other type of transformations of data. While DNN do not generally take into consideration these characteristics of the structure of the data there is a series of algorithm which do, they are called Convolutional Neural Networks (CNN).
Invented by LeCun *et al.* [38] CNN are neural networks which respect the locality of the input data and are translationally invariant [8].
The general structure of a convolutional algorithm stems from the architecture of a neural network. So it is divided in layers connected through weights. The difference is in that it is made up of two main operations: discrete convolution and pooling. These operation reduce the dimension of the input and are repeated through the network. The final output is usually fed to a DNN in order to make a classification, for example. Figure 2.5 gives a visual idea of the steps followed by a CNN.
The core idea of CNN's is to use a Local Receptive Field (LRF). That means pro-

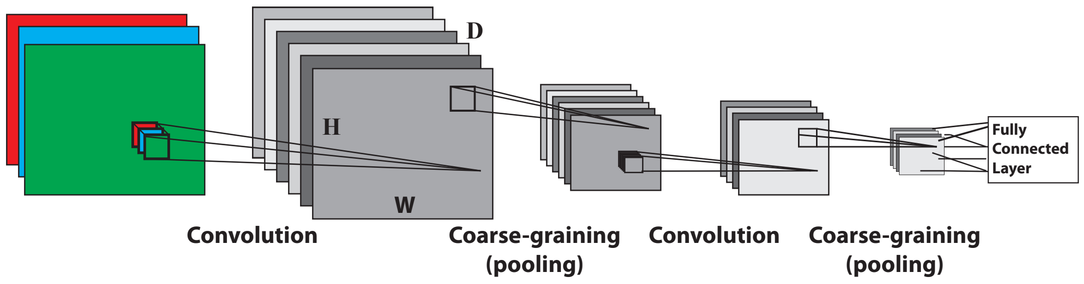


Figure 2.5: Architecture of a Convolutional Neural Network. The input on the left is a three dimensional tensor consisting of three images in RGB colors and thus three channels (see main text). The algorithm proceeds repeating twice a two dimensional convolution and a pooling. The first convolution has D filters applied to all the three RGB channels producing D channels of reduced height H and width W. The pooling clearly reduces intensively the layer dimension. The result is fed to a fully connected neural network. The picture is taken from [8].

cessing the data in patches which preserve the structure of the input. This is done by applying a filter, or kernel, which is a matrix (or tensor in more dimension) of weights that slides across the input. A kernel performs the same affine transformation that a DNN would do but it does it considering only a small portion of the input at a time and using the same weights across the whole input. That is commonly referred to as the *shared weights* advantage of CNN. In this way we can expect a kernel to learn a specific feature of the data and concentrate to recognize only it. This first operation is a discrete convolution. Although CNNs can be applied to all

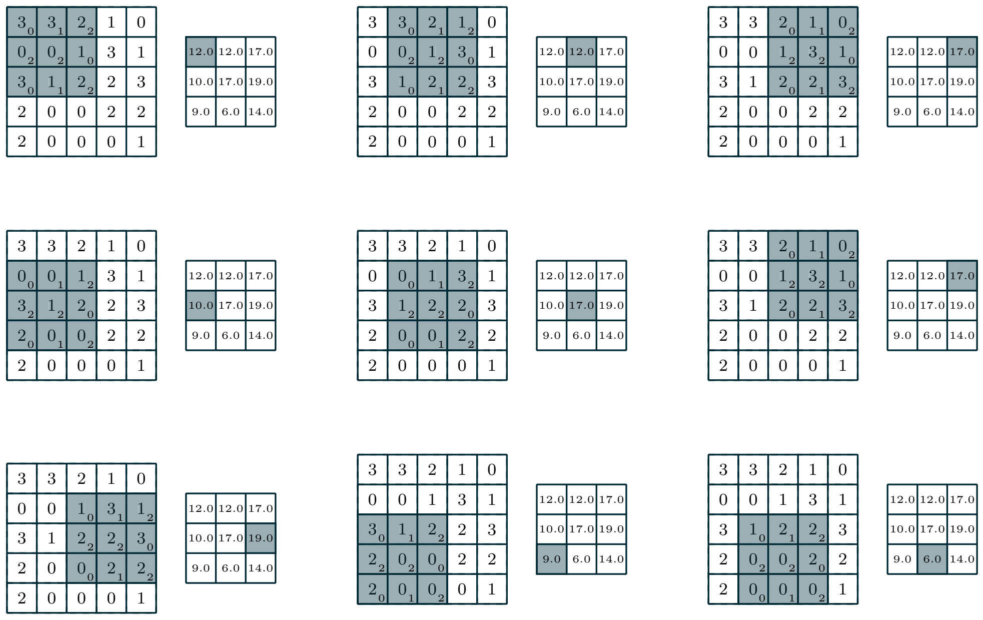


Figure 2.6: Discrete convolution of a 3 × 3 kernel across a 5 × 5 input image. Each of the nine weights of the filter multiplies a different input value (in blue) and then they are all summed up to form the receptive field (in green). In this context no bias was added to the convolution, Image taken from [39].

kinds of dimensions we will stick to 2D inputs. Let us revise the main parameters of this algorithm:

- **Patch-size**. The dimension of the kernel decides the new dimension of the receptive field. We denote it as L_{patch}.
- **Padding**. This technique overcomes the problem of the side values that are considered a smaller number of times in the convolution operation. It consists in adding N_z zeroes to the corner of the image.
- **Stride**. It is the "walk" that we allow the filters to use. In the case of Figure 2.6 the stride is one both along the x and y dimensions. In two dimensions is indicated by (s, s).
- **Channels**. The input of a CNN is a N-dimensional tensor, the first N-1 dimensions represent the data type (a vector, a 2D image, a 3D image *etc.*), the last dimension is the number of *channels*, N_C. That is due to the fact that in data science we often work with images with RGB colors and there needs

to be a matrix of pixel for each one of them. So the input to a CNN that classifies, for example, 28x28 images can be a tensor of dimension (28, 28, 3). The kernels thus obtain a new dimension themselves: $(L_{patch}, L_{patch}, N_C)$ so they add the contributions of all the channels together.

- **Number of filters**. Lastly, one always want to consider applying more then one filter, in order to learn more local features of the data. Thus the same convolution of a kernel will be repeated by N_f other kernels, each one with their shared weights.

Figure 2.6 can help understand how this works. The picture shows the single steps of convolution made with the kernel: One can verify by direct calculation that for

$$\begin{array}{|c|c|c|}\hline 0 & 1 & 2 \\ \hline 2 & 2 & 0 \\ \hline 0 & 1 & 2 \\ \hline \end{array}$$

each 3×3 sub-set of the input 5×5 matrix, there is a element wise moltiplication with the values of the filter. In the particular example of Figure 2.6 there is no padding, the stride is $(1, 1)$, there is only one channel (one input matrix) and only one filter (the single kernel) which obviously has patch size $(3, 3)$.
All in all, if a squared 2D input $a^{(0)}$ has dimensions $L \times L$, we can expect the first layer of the CNN, the receptive field, to have $a^{(1)} = L_1 \times L_1$ neurons with

$$L_1 = \frac{L - L_{patch} + 2N_z}{s} + 1 \tag{2.39}$$

Actually, there is going to be one receptive field $a^{(1)}$ for every filter used. The general rule to calculate the activation of a layer (l), from layer $(l-1)$ when applying a convolution of N_f filters with size $L_{patch} \times L_{patch} \times N_c$ is then

$$a^{(l)}_{x,y,k} = g_l \left(\sum_{m=1}^{L_{patch}} \sum_{n=1}^{L_{patch}} \sum_{c=1}^{N_C} a^{(l-1)}_{x+m-1,y+n-1,c} W^{(l)}_{mnck} + b^{(l)}_k \right) \tag{2.40}$$

where x, y are the coordinates of the datapoints,$<$ along which the weights W act, c is summed over all the N_c channels of the previous layer and k runs over the number of filters we wish to have for the next layer. Thus at each layer the information of all the channels of the previous layer is summed up by every filter, each one of those will produce a new channel itself.

The shared weights nature of this algorithm drastically reduces the number of the network parameters. A second operation is often added to convolution and it is a "coarse-graining" procedure which is necessary when dealing with very large image inputs (that is why it will not interest us). This operation is called pooling and it consists in reducing the dimension of the receptive field by sliding through it and assign a single number to every patch. The two mostly used way to do this are maxpooling, where the largest element in every patch is considered and average pooling which takes an average of the values of a patch. The result is pretty similar to that of a step of the lattice renormalization flow.
The pooling operation clearly loses much of the information and it is thus suggested for data with high correlation, meaning redundant information.

2.2.3 Principal Component Analysis (PCA)

A central aspect of Condensed Matter physics is dimensional reduction. That is because typical problems described by quantum many body theory are subject to the curse of dimensionality, that is the exponential growth of a problem with the number of degrees of freedom.
In data science, and in physics, dimensional reduction can help understanding better the underlying characteristics of a dataset and thus apply the best algorithm to make predictions.
The reduction of dimensionality of a problem is also a common theoretical approach in physics. Almost every statistical and field theory, for example, are usually described by an extreme or infinite amount of degrees of freedoms. Indeed, they rely on capturing the physics of the whole theory in fewer parameters such as the order parameters, which expected to act as a degree of freedom.
Reducing the dimensions of the problem can also lead to some issues when the dimensions are reduced below the number of *intrinsic* dimensions [5] (defined as the minimum dimension necessary to reproduce the data). Its result is the so called "crowding problem" for which points that are separated in the original space tend to overlap on the target, less-dimensional, space.
Principal Component Analysis (PCA) is a statistical method for dimensional reduction that lies on the assumption that orthogonal directions of the data with the largest variance are considered more *informative.* The main idea behind this algorithm is to perform a rotation of the input space of the data in order to align to the principal axes of variance in the new rotated basis [5].

Even though PCA loses a part of the information it still can be used as a data preprocessing tool. It has led to interesting results in studying the classical 2D Ising model for detecting the phase transition (Wetzel, [40]) and finding a correspondece with the Renormalization Group flow across the phase transition [41].

Dimensional Reduction In mathematical language PCA is the Singular Value Decomposition (SVD) of the matrix of the data.
Let us assume we have N data point $\{\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_N\}$ living in a D-dimensional space. We construct a *design matrix* $\boldsymbol{X}$ as a $N \times D$ matrix where each row is made up of the D components of each datapoint and each column is considered as a feature of the data [8].
We now normalize the data allowing them to have zero mean and arbitrary variance. In this way the symmetric $D \times D$ correlation matrix is simply given by

$$\boldsymbol{\Sigma}(\boldsymbol{X}) = \frac{1}{N-1}\boldsymbol{X}^{\boldsymbol{T}}\boldsymbol{X} \tag{2.41}$$

It is important to standardize the data so that the diagonal of $\boldsymbol{\Sigma}(\boldsymbol{X})$ contains the variance of each feature and $\boldsymbol{\Sigma}(\boldsymbol{X})_{ij}$ is the covariance, or connected correlation for physicists, between features i and j.
Now we look for a decomposition of the covariance matrix that highlights the directions in which data have a larger variance and damps the redundancy between basis vectors. In order to do so we apply SVD to the design matrix, that is we look for a representation for $\boldsymbol{X}$ such that $\boldsymbol{X} = \boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{\boldsymbol{T}}$. $\boldsymbol{U}$ has as its columns the left singular vectors of $\boldsymbol{X}$, $\boldsymbol{V}$ has the right singular vectors and $\boldsymbol{S}$ has the singular values s_i of the design matrix.
Thus, we can rewrite the covariance matrix as

$$\boldsymbol{\Sigma}(\boldsymbol{X}) = \frac{1}{N-1}\boldsymbol{V}\boldsymbol{S}\boldsymbol{U}^{\boldsymbol{T}}\boldsymbol{U}\boldsymbol{S}\boldsymbol{V}^{\boldsymbol{T}} = \boldsymbol{V}\left(\frac{\boldsymbol{S}^2}{N-1}\right)\boldsymbol{V}^T \equiv \boldsymbol{V}\boldsymbol{\Lambda}\boldsymbol{V}^{\boldsymbol{T}} \tag{2.42}$$

where $\boldsymbol{\Lambda}$ is a diagonal matrix with decreasing eigenvalues. With the simple steps of (2.42) we created a bridge between the SVD of the design matrix and the eigendecomposition of the covariance matrix. In fact the singular values of $\boldsymbol{X}$ are related to the eigenvectors of $\boldsymbol{\Sigma}(\boldsymbol{X})$ by $\lambda_i = s_i^2/(N-1)$. At the same time the right singular vectors of the design matrix are the eigenvectors of the covariance matrix. In this way it is mathematically equivalent to decompose the design matrix into its singular values or to calculate the eigenvectors of the correlation matrix, in order to produce

the PCA. The diagonalization of the correlation matrix implies that the new directions (represented by the eigenvectors) we found are not correlated to each other, *i.e.* we erased completely the linear correlation between data points.
Having said that, we need now to find the projections of the original data into the new lower-dimensional space.
Firstly we select the larger eigenvalues of the correlation matrix. We do that taking into consideration the sum $\lambda_i / \sum_{J=1}^{D} \lambda_j$. This term is the percentage of *explained variance*, basically a quantity that states the amount of information contained by each new principal component.
Now we select the eigenvectors in $\boldsymbol{V}$ corresponding to the p largest eigenvalues we selected and acting with

$$\boldsymbol{Y} = \boldsymbol{V^T X} \tag{2.43}$$

we obtain the p-dimensional representations of our dataset which contain an amount of information of the original data equal to the explained variance.
Clearly, low values of the explained variance suggest that the intrinsic dimension of the data might be larger the expected and thus it require the calculation of more principal components.
We will see an example of the implementation of this algorithm in the analysis of the data of the non-interacting Kitaev model in Chapter 3.

2.3 Deep Learning in Condensed Matter Physics

In the previous sections we tried to give an idea of what are the main features behind Machine Learning and Deep Learning algorithms and how they can be exploit in a physics oriented fashion.
The part of physics we are interested on working on is Condensed Matter and the study of topological phase transitions, specifically for the Kitaev model. Chapter 3 is entirely focused on the presentation of a work in this field. We keep this last section as a presentation of two examples of application of Deep Learning to the study of phase transitions.
In the first example we build a neural network to study the phase transition of the classical Ising 2D Model. The second one is based on the reproduction of interesting results obtained by Nieuwenburg *et al.* by applying what they call as *confusion scheme* to study in an unsupervised fashion to the topological phase transition of the Kitaev model with nearest neighbor coupling in one dimension.

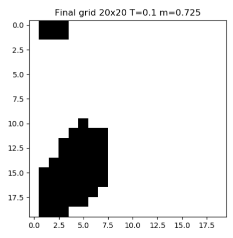
2.3.1 Classifying Phases of the Ising 2D Model

The classical Ising model in two dimensions is a very simple model which shows a phase transition.
This basic model is very popular thanks to its simplicity and to the non trivial fact that it shows a phase transition. This is why it is always studied as a benchmark for new methods, in particular in Machine Learning.
It describes an interaction between classical spin degrees of freedom S_i which can only take values $+1, -1$. Let us consider a two dimensional Ising model on a square lattice. If we only consider nearest neighbors coupling and no external magnetic field the Hamiltonian is quite simple

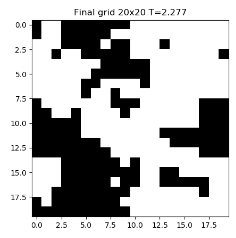
$$H = -J \sum_{\langle i,j \rangle} S_i S_j \tag{2.44}$$

where $J > 0$ quantifies the interaction between neighbouring spins and the sum runs on the four nearest neighbors of every lattice site. The presence of a minus sign characterize the groundstate of the model as *ferromagnetic* because all the spins line up in the same direction in order to minimize the total energy. The groundstate is reached for low temperatures, while in the limit of high temperatures all the spins randomly point either up or down (*paramagnetic phase*). The transition between ferromagnetic and paramagnetic phase occurs when the $\mathbb{Z}_2$ symmetry of the paramagnetic phase is broken down at low temperature. This phenomenon occurs at the critical temperature $T_c/J = 2/\ln\left(1+\sqrt{2}\right) \approx 2.269$ [42].
In this section we recreate the results of the paper by Carrasquilla *et al.* [12] where they apply a DNN to the study of the Ising 2D model, along with a CNN to study the Ising 2D lattice gauge theory, to show the basic feature of a machine learning approach to study phase transitions. We will only focus on the first part.
The purpose of our work is to show how a neural network can be trained for a simple binary classification, thus we will not dive into too many details of the training and testing.
Let us quickly present the architecture of the model

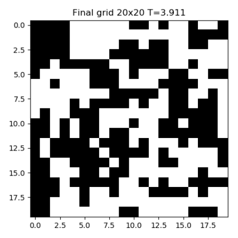
- **The dataset**. We generated 5000 binary spins configurations of $L \times L$ lattices with $L = 10, 20, 30, 40$ and 50. We divide it into training, validation proportion $3 : 1 : 1$. The configurations were generated with a Monte Carlo Markov Chain for N^2 steps at 500 different temperatures in the range 0.1 to 5 taking $J = 1$. Some configurations are shown in 2.7. This was implemented as a



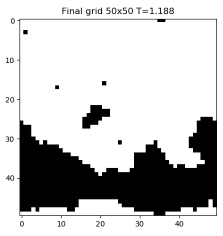
(a) L = 20, T = 0.1



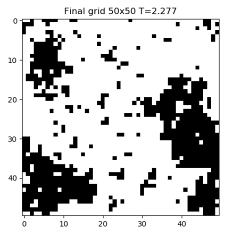
(b) L = 20, T = 2.277



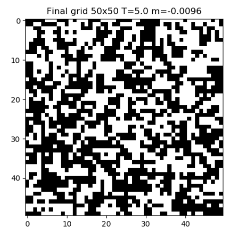
(c) L = 20, T = 3.911



(d) L = 50, T = 1.188



(e) L = 50, T = 2.277



(f) L = 50, T = 5.0

Figure 2.7: Ising 2D spin configurations at different temperatures and for lattice size $L = 20$ (first row) and $L = 50$ (second row). The three temperatures considered are, in order from the left, lower, close and larger then the critical temperature $T_C = 2/\ln\big(1+\sqrt{2}\big)$.

python program. It needs to be taken into consideration that for low temperatures there some domains form which are hard to break down and thus their magnetization is not zero. This will affect the precision of the network.

- **The algorithm**. We created a neural network with input a L^2 array of binary spins $+1, -1$, one hidden layer of 100 neurons and two output layers. To do this we programmed with Keras [43], a high level API for development of neural networks which uses TensorFlow [44] as Backend. The activation in the first layer is a ReLU, for the output is a sigmoid function. The model has a total of $(L^2+1)\times 100+(100+1)\times(2)$ parameters between weights and biases. Thus it goes from having 10,302 parameters for $L = 10$ to 250,302 for $L = 50$.

- **The training**. The gradient descent is performed by the ADAM algorithm [35]. We use a cross-entropy cost function. We try different batch sizes and epochs but do not focus on regularization terms or learning rate. The accuracy of the model is simply calculated as the percentage of correctly classified samples out of the 1000 test data points.

- **The output**. The DNN has two output neurons: a "hot" neuron that should activate (give a probability close to 1) when $T < T_C$ and a "cold" neuron which behaves in a opposite way.

We consider the output of the network for $L = 20$, batch size 100 and 50 epochs of training as a reference. Varying the dimensions of the lattice, batch sizes and epochs only slightly changes the result as we will show later. The plot of the output neurons is shown in Figure 2.8. We take it as example to show the common feature with the other outputs for different L. The picture shows the output of the cold

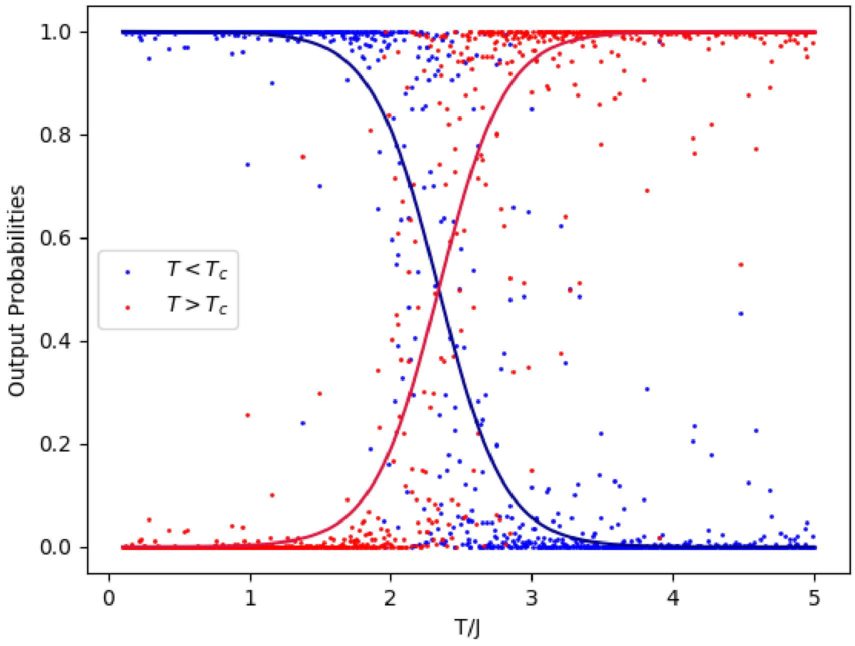


Figure 2.8: Output probabilities of the "cold" and "hot" neuron. The ferromagnetic and paramagnetic are clearly distinguished by the algorithm. The DNN looses precision around the critical temperature because it has no reference on how to separate the two phases. The picture shows the classification of the whole test set of 1000 data points.

neuron in blue and the hot neuron in red. The scatter plot is fitted with a sigmoid function and the crossing point of the lines is at $T = 2.285$, which can be considered as the critical temperature "learned" by the model. We notice that the network is not precise in the distinction of phases around the critical temperature. That is understandable since the difference in the spins configurations around T_C are not neat at all even at eye sight.

There is a discrepancy in the number of incorrectly classified points. The right side

	$T < T_C$	$T > T_C$
$T < T_C$	436	6
$T > T_C$	54	504

Table 2.1: Confusion matrix of the output of the DNN with accuracy 94%. It shows the number of predicted values (rows) against the expected ones (columns). The off-diagonal values represent the mistakes of the network. The number of points over the critical temperature mis-classified as ferromagneticly ordered is much larger then the other wrong points. The same situation arises for every configuration tried.

of the graph, indeed, shows more errors. That is probably due to the fact that the total number of up or down spins in the disordered phase is often similar to the cases of low temperature where a domain wall forms (see Figure 2.7). The accuracy reached is 94% and the confusion matrix is presented in the table 2.1 The confusion matrix shows the unbalance we were mentioning more clearly, this behaviour repeats for different sizes of the configurations. So we can infer that the neural network might be discriminating the two phases by a mere count of the up and down spins. The algorithm is in fact incapable of recognising explicitly visual patterns such as the formation of a domain wall. In order to exploit this and other symmetry properties there would probably need a convolutional neural network.
Since neural networks usually feed on large amounts of data we try to reproduce the outcomes doubling the dataset size. Thus using a total of 10 thousand configurations, 1000 of which for validation and 1000 for testing.
The plot of the outcome is similar to the one for fewer data but we do see a 1.5% increase in the precision on the test set for both $L = 20$ and $L = 30$.
This first basic example permits us to gain more intuition in the functioning of a neural network. In Chapter 3 we will exploit this knowledge and compare different sizes of datasets, consider the convergence of the loss function in the training and be more precise under different aspects.

2.3.2 Learning the Kitaev Model by Confusion

In this section we present a particular technique invented by van Nieuwenburg *et al.* [13] called "confusion scheme". This technique creates an unsupervised model through the use of neural networks, which are supervised algorithms, to study the topological phase transition of the Kitaev model. It is very interesting to study because unsupervised approaches are key to investigate unknown data. One thing

is to "teach" a network how to recognise phases of matter, as we did in the last section, by telling it exactly which is one. Another thing is creating an algorithm that can tell you where a phase transition happens without telling it where it should find it.
The idea behind the confusion learning is simple yet effective. It consists in training many neural networks with the same data but different labels. Let us assume the data depends on some parameter, in the case of the Kitaev model we consider the chemical potential μ. Let us also assume the model goes through a phase transition for a specific value of μ that we will call μ_c. Thus a neural network would have the data labelled as 0 or 1 according to μ which generated it if it is larger or lower then μ_c. This is standard procedure for training a DNN to recognise a phase transition. Now, let us consider the case in which the critical value μ_c is not known and it is only supposed to reside inside an interval of possible values $[\mu_{min}, \mu_{max}]$. One can consider N values of μ inside that interval, one of which should correspond to the real critical value μ_c (or should be close enough, the precision can be set by choosing a large N). The "confusion" is introduced when training N different neural networks with the same architecture each one with a different value μ_i to discriminate the two phases. Thus, for example, the network with $\mu_i = \mu_{min}$ will have all its data labelled as 1 (because they are all above the critical point), while the network with $\mu_i = \mu_{max}$ will have all its data labelled with 0, for obvious reason. The confusion is due to the wrong labelling of the data, of course only the network trained with the real, unknown, value $\mu_i = \mu_c$ will have a sensible labelling. This intuitively means that the accuracy of this network will be maximum, or at least larger then the accuracy of the other networks since they are working with wrong data. In particular if there is any characteristic that actually separates the data of the two phases, only the network with the right μ_c will be able to classify them. Interestingly enough we can predict the plot of the accuracies of all the networks to have a "W" shape. That is due to the fact that both the example of $\mu_i = \mu_{min}$ and $\mu_i = \mu_{max}$ will be trained to assign the same label to all the data, and will thus perform perfectly (with 100%) on new data. When changing the μ from the extreme values we expect the accuracy to decrease and then to increase again close to the right values of μ_c. This same reasoning can be applied to larger intervals of the parameter, at bigger computational cost, basically being able to predict any phase transition or even multiple ones which will be all recognisable due to a peak in the accuracy of the networks [13].
We now reproduce the same results of van Nieuwenburg *et al.*. We consider the

Kitaev chain which shows a topological phase transition at $|\mu| = 2|t|$. We firstly introduce the neural network we used for classification before applying the confusion scheme. The data we worked with are the 10 largest eigenvalues of the Entanglement Spectrum (ES). This data was calculated in [45] by applying a bipartition of the chain into subchains A and B and calculating the partial density matrix by tracing out one of the two sublattices, *i.e.* $\rho_A = \mathrm{Tr}_B |\psi\rangle\langle\psi|$. We call the eigenvalues of ρ_A as r_i and the ES is given by the values $\lambda_i = -\ln r_i$. The degeneracy of the groundstate in the topological phase is reflected by the same degeneracy in the spectrum.

- **Dataset**. We used the 10 largest values of the Entanglement Spectrum calculated at 4000 different values of μ from -4.0 to 0. So the design matrix is a 10×4000 matrix. The first four eigenvalues for different μ/t values are shown in Figure 2.9. The other eigenvalues are all close zero.

- **Architecture**. For this classification we considered a DNN with an input of 10 values, a hidden layer of 80 neurons with sigmoid activation and two output neurons with sigmoid activation showing either topological or trivial phase.

- **Training**. We followed the same approach as the Ising2D model so ADAM optimizer for gradient descent and cross entropy validation.

- **Output**. The output was produced by testing the algorithm on 1000 samples. The neurons output the probability of a sample to belong to either the topological or the trivial phase. It is shown in figure 2.10

The output classification of the neural network is shown in Figure 2.10. The degeneracy of the spectrum makes this kind of classification particularly easy. The network reaches an accuracy of 100% in a few epochs. From the image we can see that the precision of the neurons follows the values of the spectrum because it is probably learning that their values are the indicator to learn to discriminate the two phases, along with the already mentioned degeneracy. We now turn to the application of the confusion scheme. We consider a window of values for the chemical potential $-4 < \mu < 0$ and train 20 DNN assuming that their value of μ is the critical μ. The results of the different trainings are presented in Figure 2.11. The expected "W" shape is obtained. As predicted, the networks trained with the extremal values of μ have a 100% accuracy. The precision then linearly decreases with rising μ and it reaches a peak for $\mu = 2$ where we know the phase transition occurs. This very simple model has a very intuitive approach but it relies on a very interesting

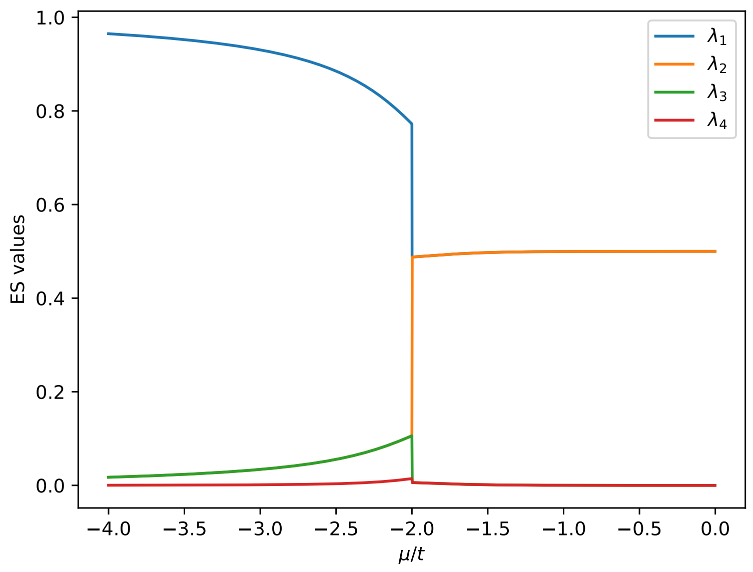


Figure 2.9: Entanglement Spectrum of the Kitaev chain. The first 4 elements of the spectrum are shown. When $\mu/t \in [-4,-2]$ the eigenvalues λ_2 and λ_3 overlap. The degeneracy of the ES is evident in the interval $\mu/t \in [-2,0]$ where the two largest eigenvalues λ_1 and λ_2 overlap as much as λ_3 and λ_4.

reasoning. If we actually applied this method to learn the phase transition of a model with unknown phase diagram we might learn the presence of a transition due to some change in the data. That it precisely what was done in paper [13] where they applied another type of confusion scheme in learning the phase diagram of the random filed Heisenberg chain.

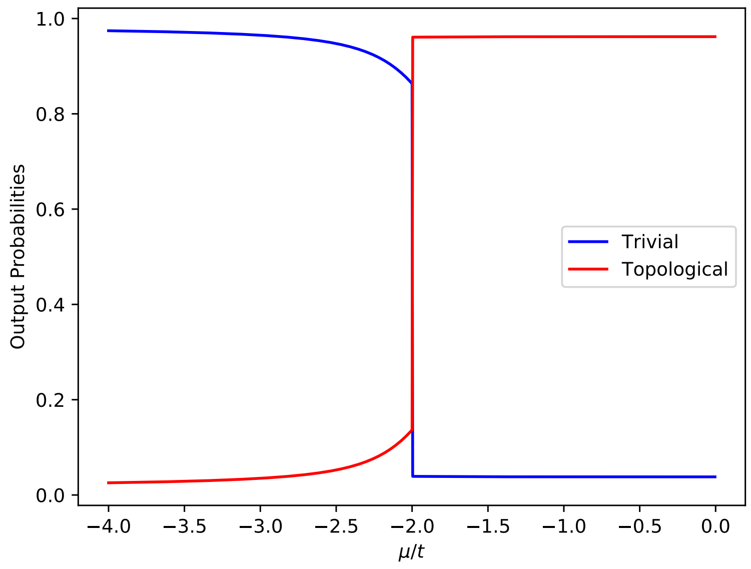


Figure 2.10: Output neurons probabilities. The plot shows the probability after the sigmoid activation of the two neurons of the DNN. One is supposed to guess the topological phase and the other one the trivial.

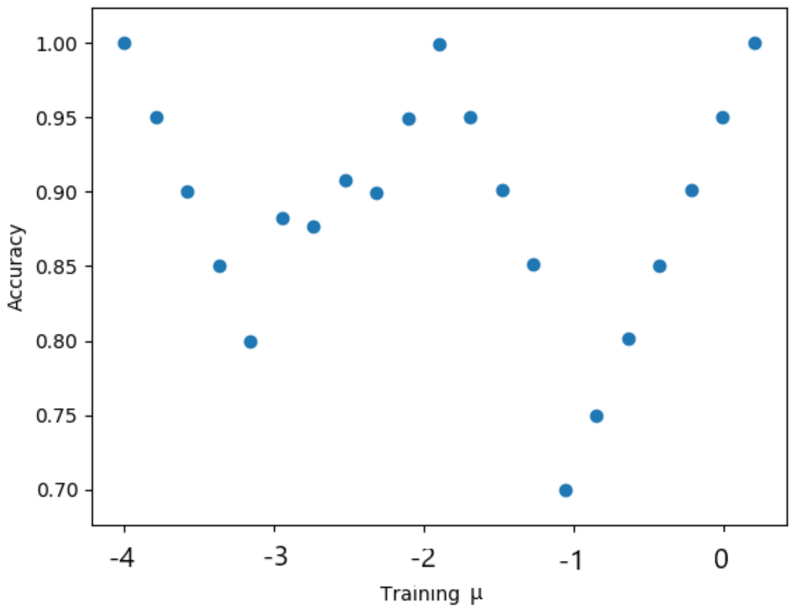


Figure 2.11: Plot of the accuracies of the different 20 neural networks trained with "confused" data. The "W" shape lets us infer there is a phase transition at $\mu = 2$

3 Results

As it is customary in every Machine Learning procedure let us start by introducing the problem, in order to devise the most suited computational model for it. As already mentioned in the first chapter, the interacting Kitaev model is not described by a quadratic Hamiltonian. Thus, it cannot be diagonalized and we do not have analytical results that allow us to build its phase diagram. Nonetheless, we have analytical results for the non-interacting Kitaev model with nearest and next to nearest neighbour coupling as shown in sections 1.4 and 1.5.

Our main goal is to be able to predict the phase diagram of the interacting model by using data of the non-interacting model (from now on we will assume that non-interacting and interacting data means data generated for the next to nearest neighbor and interacting Kitaev chain). That requires training and testing an algorithm on non-interacting data to see if it learns their topological phase. Then, applying it to a different test set which is made only of data obtained from the interacting model and thus different from the ones the network was previously trained and tested on. This is, on a physical principle, an extremely interesting goal. Being able to make any sort of predictions about a model for which we do not have analytical results is of primary importance in the field of Condensed Matter Physics. On the other hand, on a machine learning research point of view, we are asking for a very particular type of generalization. That is, we are not only going to test our algorithms on the data they were trained on. We are also going to use as test set data which are in the same topological phases of the training data but still belong to a different model. For this reason, the core idea behind these results is not only to find a ML algorithm capable of making predictions about non-interacting data but also to understand how that is possible and what are the "features" that the algorithm learns on non-interacting data to predict interacting ones.

For the reasons listed above, this chapter is not an example of application of DL to study the efficiency of algorithms. It is focused on describing physics through the models.
We will start presenting the task in a more detailed way: showing the structure of the data we work on and the measures that needs to be taken.
After that we will apply a preprocessing procedure to have more insight on the data we use. That is, the Principal Component Analysis presented in section 2.2.3.
Then, we will show the main results of applying a fully connected neural network trained on the non-interacting data to classify the topological phases of the interacting data. We will open up the computational "blackbox" in order to understand why it fails at recognizing parts of the test set.
Eventually, we will train a Convolutional Neural Network on the same non-interacting data and see that it is able to predict the whole phase diagram of the interacting model, contrarily to the Deep Neural Network. Also in this case we will open up the network and show into details what features are learnt.

3.1 Setting Up the Procedure

In the introduction we outlined the goal of this chapter and how to reach it. First thing of setting up a Machine Learning problem is to decide the data to work on, the model and the cost function.

- **The dataset**. The correlation functions, or structure factors, of the non-interacting Kitaev model with next to nearest neighbor coupling. Labelled with the topological phase they were calculated in.
- **The model**. There is no known function which is able to predict the phase of a set of correlation functions. Given also the complexity of the generalization required (from non-interacting to interacting data) we decide to implement a neural network.
- **The cost function**. Since we are dealing with a classification process we will use cross entropy to evaluate the performance of our networks.

3.1.1 Non-Interacting Data

One of the main ways to study the Kitaev model (1.53) is to measure its correlation functions: the one body operators $C(j,l) = \langle c_j^\dagger c_l \rangle$ and $F(j,l) = \langle c_j^\dagger c_l^\dagger \rangle$. They can

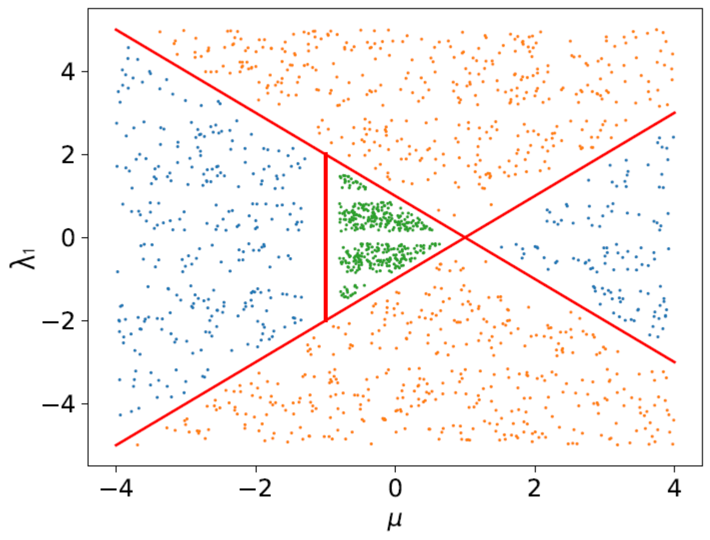


Figure 3.1: Reconstructed phase diagram of the next to nearest neighbor model. Parameters λ_1 and μ vary in the interval $[-5, 5]$, $[-4, 4]$. The three topological phases are identified by colors. Blue is for 0 winding number, orange for ± 1 and green for 2. The diagram reproduces the phase diagram of the NNN model shown in Figure 1.3 but we are varying different parameters so the critical lines are slightly different. Notice how there is only one sector for $w = 2$, and two sectors for $w = \pm 1$ and $w = 0$. The white lines

be calculated exactly over the fermionic vacuum for the sites j, l of a L long chain for both the NN (nearest neighbor) and NNN (next to nearest neighbor) models as shown in Chapter 1. More precisely, we usually work on their Fourier transforms:

$$C(k) = \frac{1}{\sqrt{L}} \sum_{j,l} e^{i(j-l)k} C(j,l), \qquad F(k) = \frac{1}{\sqrt{L}} \sum_{j,l} e^{i(j-l)k} F(j,l). \tag{3.1}$$

Throughout the whole chapter we work with a Kitaev chain of $L = 100$ sites. The analytical structure of the correlation functions was already calculated in the first chapter, we rewrite it for brevity:

$$\begin{aligned} C(k) &= \frac{\mu - \lambda_1 \cos k - \lambda_2 \cos 2k}{2\varepsilon(k)} - \frac{1}{2} \\ F(k) &= -\frac{1}{2} \frac{\lambda_1 \sin k + \lambda_2 \sin 2k}{\varepsilon(k)} \end{aligned} \tag{3.2}$$

along with the Hamiltonian of the model:

$$H_1 = \frac{\lambda_1}{2}\sum_{j=1}^{N-1}(c_j^\dagger c_{j+1} + c_j^\dagger c_{j+1}^\dagger + h.c.) \tag{3.3}$$

$$H_2 = \frac{\lambda_2}{2}\sum_{j=1}^{N-1}(c_j^\dagger c_{j+2} + c_j^\dagger c_{j+2}^\dagger + h.c.) \tag{3.4}$$

$$H_\mu = -\mu\sum_{j=1}^{N}(c_j^\dagger c_j - \frac{1}{2}) \tag{3.5}$$

$$H = H_\mu + H_1 + H_2 \tag{3.6}$$

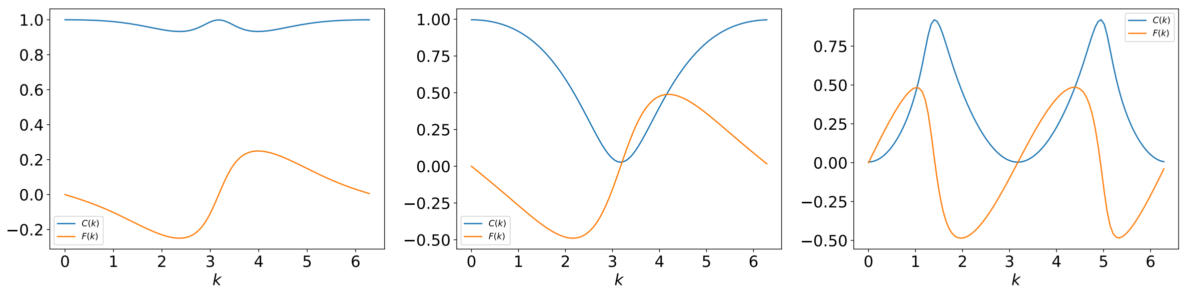
where the hopping parameter t_1 and the superconducting gap Δ_1 were both set equal to $\lambda_1 = t_1 = \Delta_1$ for the nearest neighbor interaction, same for λ_2 with next to nearest coupling.
To generate different samples we set $\lambda_2 = 1$ and vary λ_1 and μ in different intervals such as $[-5, 5]$ and $[-4, 4]$ for Figure 3.1. In this way we span all the three topological phases with winding numbers 0, ± 1, and 2. Figure 3.1 represents the datapoints generated, each dot corresponds to the values of μ and λ_1 for which we calculated the functions $C(k)$ and $F(k)$. The white lines correspond to the data used for testing and thus neglected in training. Since we took $\lambda_2 = 1$ the gap closing lines occur for

$$\lambda_1 = \pm(1-\mu), \qquad\qquad \mu = -1, \tag{3.7}$$

a direct comparison with graph 1.3 demonstrates it is the same phase diagram.
The plot of three samples generated in the three different topological sectors is shown in Figure 3.2.
We created three datasets of different size to train the ML algorithms. We did that in order to show, if necessary, the difference in working with larger or smaller amount of data. We generated

- **Dataset** $\mathcal{D}_1$. With a total of 5000 samples: 4000 for training, 500 for validation and 500 for testing.
- **Dataset** $\mathcal{D}_2$. With a total of 10 000 samples: 8000 for training, 1000 for validation and 1000 for testing.
- **Dataset** $\mathcal{D}_3$. With a total of 25 000 samples: 20 000 for training, 2500 for validation and 2500 for testing.

Every dataset consists of 2/5 of samples generated at winnding number 0, 2/5 at winding number 1 and 1/5 at winding number 2. Throughout the chapter we will simply refer to dataset $\mathcal{D}_1$, $\mathcal{D}_2$ and $\mathcal{D}_3$ assuming validation and test size are already set, unless differently specified.



(a) $w = 0$ (b) $w = 1$ (c) $w = 2$

Figure 3.2: Plots of the structure factors $C(k)$ and $F(k)$ for three different combinations of λ_1, μ. The correlations function are calculated, from the left at $\lambda_1 = -0.40$, $\mu = 0.64$, $\lambda_1 = 3.47$ $(w = 0)$, $\mu = -7.00$ and $\lambda_1 = 13.69$ $(w = 1)$, $\mu = 8.80$ $(w = 2)$.

3.1.2 Interacting Data

As already mentioned the interacting data for the model (1.133) were obtained with the DMRG. That is due to the fact that there are no analytical solutions for the model. To work on the interacting Kitaev chain we produced $C(k)$ and $F(k)$ for the different values of μ and V in the phase diagram 3.3 which we replot here for brevity. The phase diagram shows the three phases: trivial and non trivial superconducting, Density Wave (DW). The Kitaev data can be seen as residing in only two topological sectors: the trivial phase labelled with 0 (trivial superconducting and DW) and the superconducting phase labelled with 1. The labels correspond to the winding number as defined for the non-interacting Kitaev Model.
We show the structure factors for three values of the potential to show their behaviour around the phase transition and far from it.
The first line of Figure 3.4 shows data for $V = 6$ that are all in the trivial phase with winding 0. That is the same for the third line of plots generated for $V = -2$. Both this sets of data are in the same trivial phase but they are rather different. In particular the plots 3.4g, 3.4h and 3.4i resemble the data generated in the non-interacting case at winding number 0 as in Figure 3.2a, while 3.4a, 3.4b and 3.4c slightly resemble data in Figure 3.2b with the exception that the $C(k)$ and $F(k)$

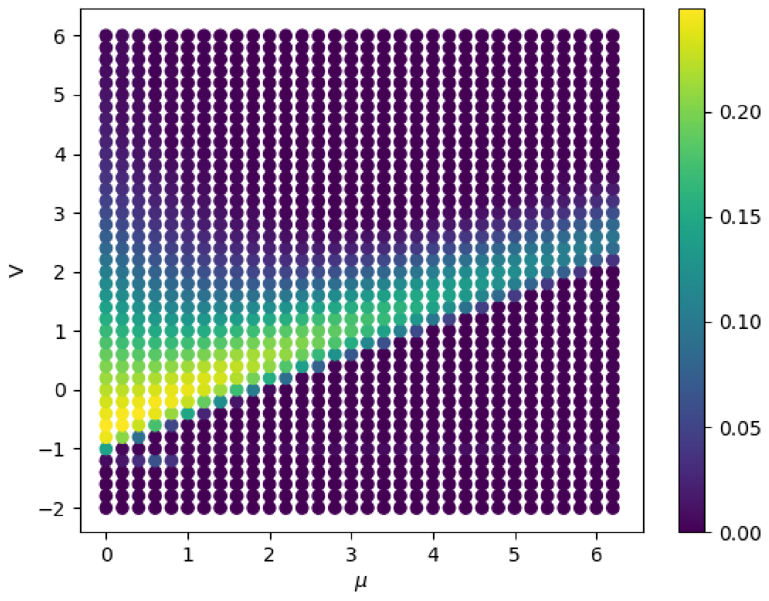
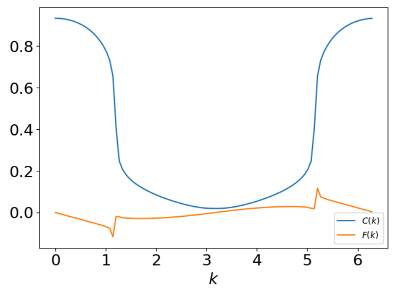
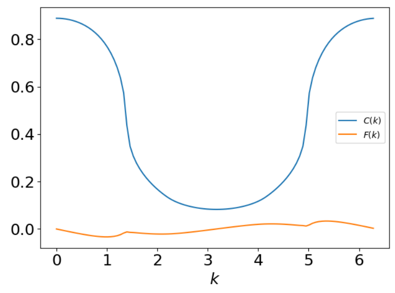


Figure 3.3: Phase diagram of the interacting Kitaev chain characterized by two topological sectors: a trivial phase for low and high values of the potentials corresponding to trivial superconducting and Density Wave phase (blue), a superconducting non trivial phase (yellow). The colorbar shows the density of state of the edge modes.

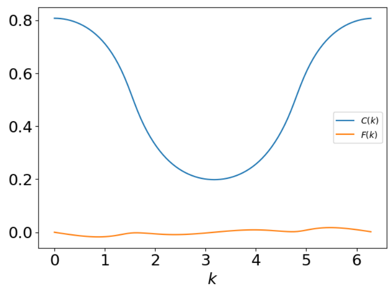
functions do not intercept.
These visual features can already make us predict that it is probably going to be more difficult to distinguish the trivial phase of the data at large V than the values at low V.
On the other hand, plots 3.4d, 3.4e and 3.4f clearly show that the system underwent a phase transition passing from correlation functions identical to the non-interacting ones at phase 1 to phase 0. In particular the correlations shownsn in Figure 3.4e are close to the phase transition and will thus be probably hard to classify.
These preliminary considerations we are making can perhaps be better understood by plotting how the correlation functions change along the μ axis for a fixed value of V.
This is done in Figure 3.5. This last series of plots shows the correlation functions for μ in the range $[0, 6.2]$ for interesting fixed values of the interaction potential V. With these plots it is easier to see what changes when the phase transition occurs, for example in Figures 3.5b and 3.5b.



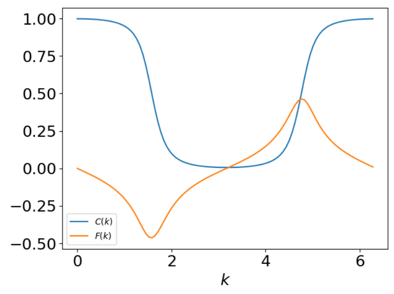
(a) V = 6, $\mu = 0.8$



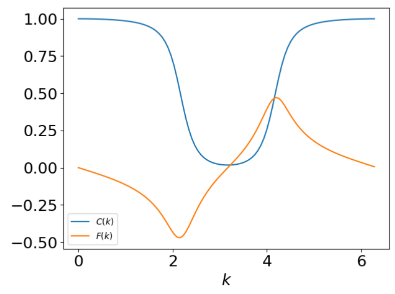
(b) V = 6, $\mu = 2.4$



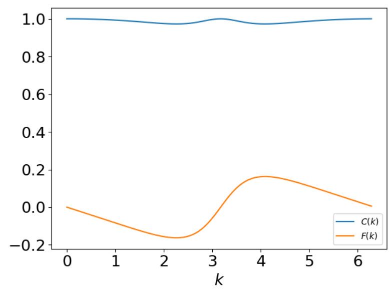
(c) V = 6, $\mu = 4.8$



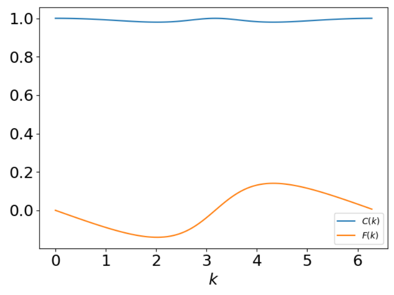
(d) V = 0.8, $\mu = 0.8$



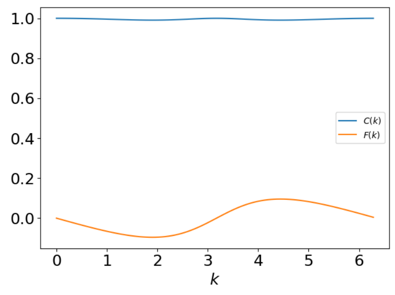
(e) V = 0.8, $\mu = 2.4$



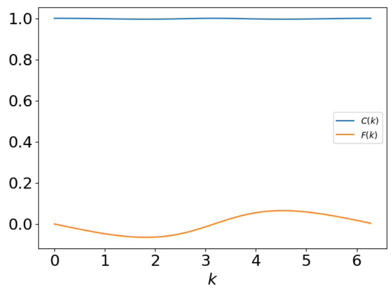
(f) V = 0.8, $\mu = 4.8$



(g) V = -2, $\mu = 0.8$

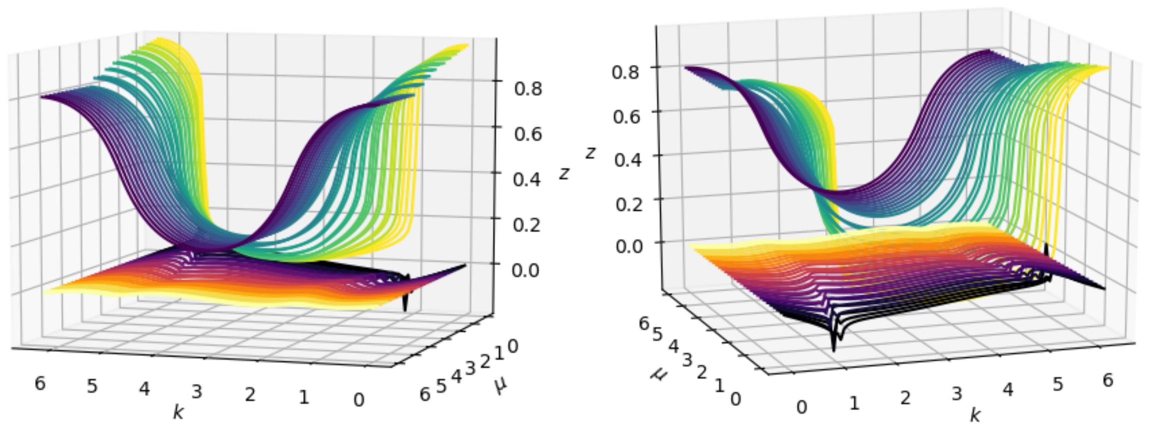


(h) V = -2, $\mu = 2.4$

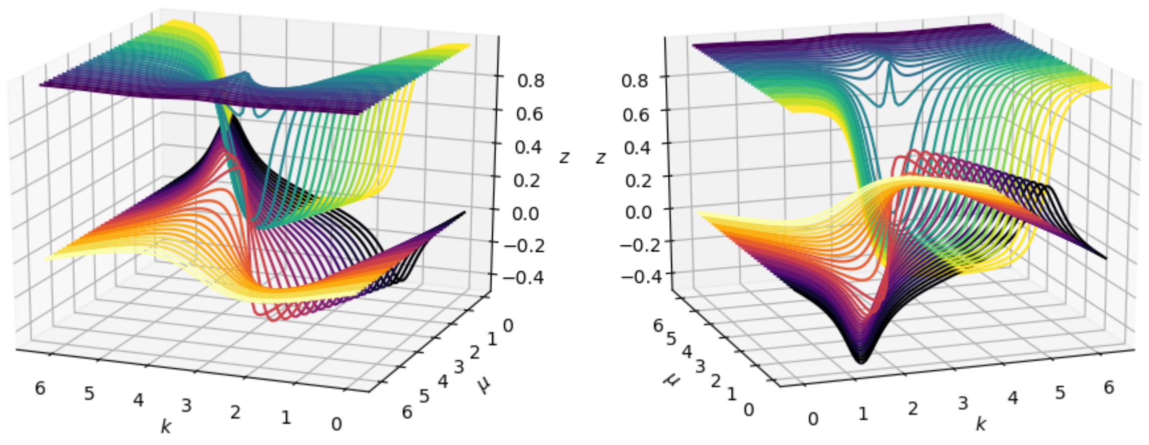


(i) V = -2, $\mu = 4.8$

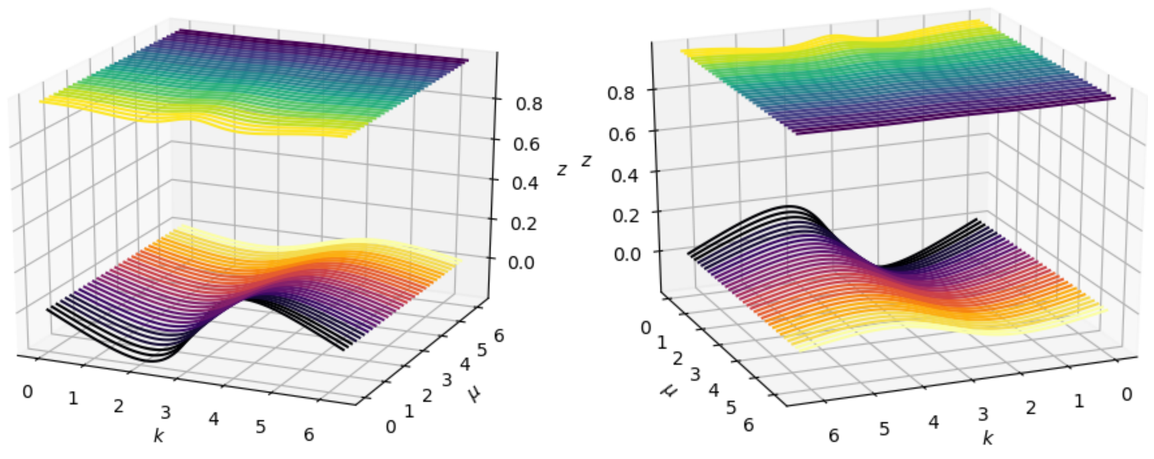
Figure 3.4: Correlation functions for the interacting Kitaev chain. The plots show the 100 values of $C(k)$ and $F(k)$ calculated at $V = 6, 0.8, -2$ for three different values of $\mu = 0.8, 2.4, 4.8$. The second line of plots passes through a phase transition from the superconducting topological phase to the trivial phase.



(a) V = 6



(b) V = 0.8



(c) V = -2

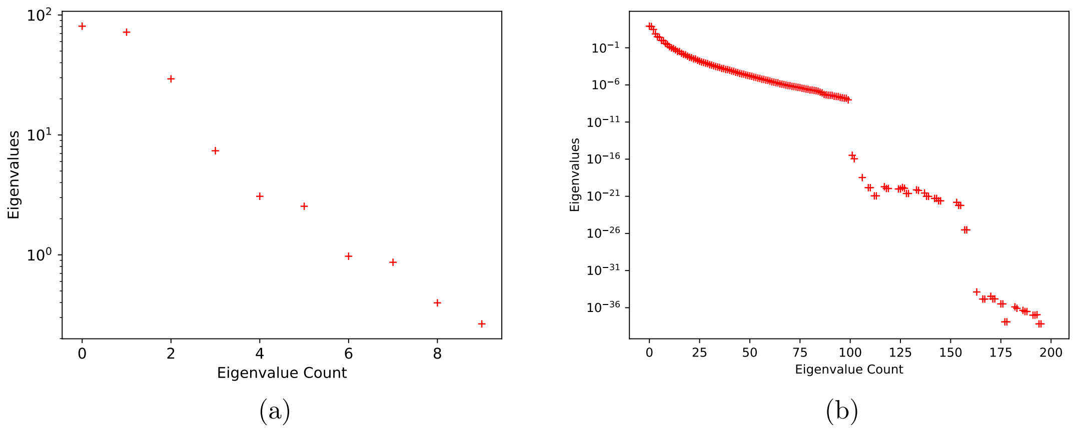
Figure 3.5: Correlation functions for the interacting Kitaev chain at different potentials V. In each graph the lines above/below are the correlations $C(k)$/ $F(k)$.

3.2 PCA of the Kitaev Model

When working on a ML project the best practice is to preprocess the data. This helps us to understand what we are working on and to extract as much information as possible from it. In this thesis we firstly analyzed our data by performing PCA. As mentioned in section 2.2.3 we can perform PCA by both doing singular value decomposition on the data or by finding the diagonal form of the correlation matrix of the data. In addition to that we can consider using the implemented methods of the package library scikit-learn [46]. Since we are dealing with a simple calculation we decide to work with the diagonalization in order to have better control on the parameters of the algorithm. Nevertheless we performed the PCA with all the three methods once to prove there were no relevant differences.

3.2.1 Non-Interacting Data.

We started by considering the non-interacting data. We selected the largest dataset $\mathcal{D}_3$ hoping to see a clustering behaviour of the datapoints with a different winding number.
The design matrix X is thus a 25000×200 matrix where each line is made up by the 100 values of $C(k)$ and $F(k)$ concatenated and it is considered as a point in a 200-dimensional space.
Interestingly, the first few eigenvalues of the correlation matrix $S = X^T X$ seem to possess large part of the information. More precisely, the explained variance (ratio of the eigenvalues to the sum of the eigenvalues) of the first three eigenvalues is 91.9% and their values are $80.7, 71.9$ and 29.4. It is true, though, that the sum of the first 6 eigenvalues adds up to 97.2% of the total information. Thus, there is some interesting information that we lose considering only the first three principal components. In fact from Figure 3.6b the eigenvalues do decrease in an exponential fashion when sorted in descending order but the first ten do not decrease so fast. Thus, we can expect to extract much of the information from the first three principal components - that is, the projection of our matrix of data onto the eigenspace of the first three eigenvectors of S - but still lose a small part of it. Let us start by projecting the data on the first two components which have explained variance of 71.1%. In Figure 3.6c we can see the results of this operation.
The graph shows there is a neat difference in between the points belonging to phase 2. On the other hand, phases 0 and 1 show a different behaviour but are still con-



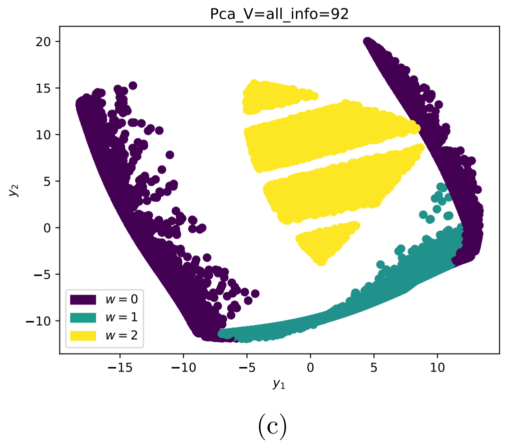


Figure 3.6: PCA of the non-interacting data. Plot (a) shows a zoom in the first 10 eigenvalues, while plot (b) on the left shows the values of all the 200 eigenvalues of the correlation matrix S, both in log scale. We can see how only the first few values contain much of the total information. The explained variance for the first three values is indeed 91.9%. The plot at the bottom (c) shows the projection of the data onto the first two eigenvectors, or principal components.

nected in some points. It is also evident that the points with winding number 2 are also separated themselves by white lines. We attribute this particular structure to the fact that when generating the data we intentionally avoided points along some lines of the phase diagram.
It is also very clear from the graph that the points of phase 2 seem to reproduce the triangular form of their phase in the phase diagram (Figure 3.2).
Lastly, even though some form of clustering seems already to appear we want to extract more information from the data and so we project the data onto the first

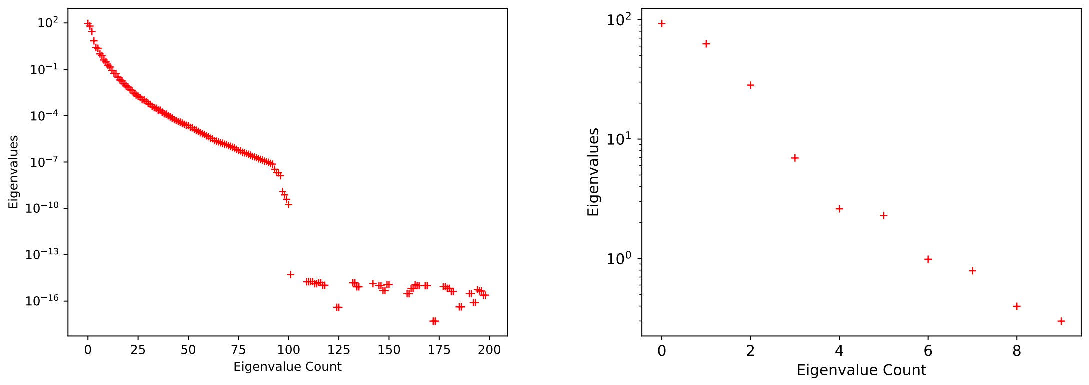
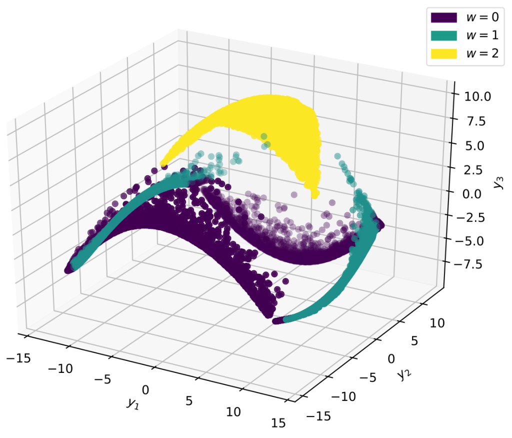


Figure 3.7: The plot shows the eigenvalues of the correlation matrix of the 20000 samples generated without obscured lines in the phase diagram. The explained variance of the first two eigenvalues is 78.3%, and of the first three is 92.5%. The general behaviour is similar to the case with the $\mathcal{D}_3$ dataset but the eigenvalues drop to zero faster.

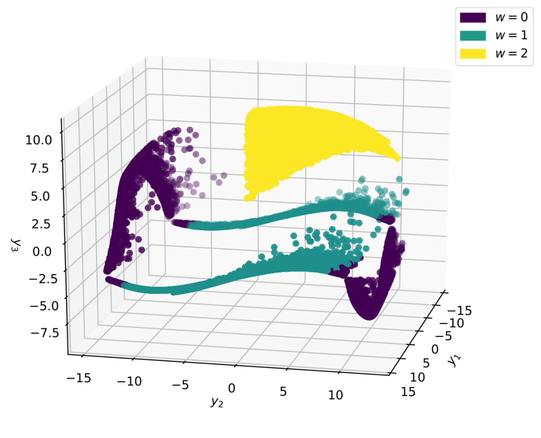
three principal components.

Putting together all these considerations we generated 20000 samples without neglecting any lines of the phase diagram and we projected along the first three eigenvectors. The log plot of the eigenvalues of this new dataset is shown in Figure 3.7.We represent the outcome of the resulting 3D plot shown in Figure 3.8 under different viewing angles. The explained variance for the first three principal components is 92.5% The first thing we notice is that the obscured lines of the phase diagram disappeared, showing that our prediction was correct. Secondly, it is clear that the PCA is able to recreate the phase diagram of the points. This is somewhat impressive considering that we only need three components, out of the total 200, of every point to separate phase 2 from the other phases. It is nonetheless predictable that the first few components can already separate our data so clearly.

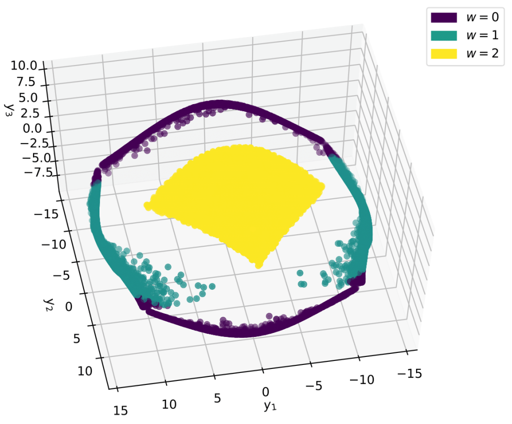
That is on one hand due to the fact that we only need two variables, that is μ and λ_1 to assign a position to the data in the phase diagram. On the other hand it is due to the fact that the correlation functions we are working on, shown in the first section of this chapter, have a high level of correlation. By that we mean that their features are redundant. For example they are are continuous. Thus, it is easy to predict the value of, say, the 100-th point of $C(k)$ if we know the first 99. Moreover they are periodic, with periodicity 2π, and thus translationally invariant in a sense that if all the input values of $C(k)$ or $F(k)$ is shifted to the left by one the function is still self-identical and an algorithm should notice that and exploit that, as we will



(a)



(b)



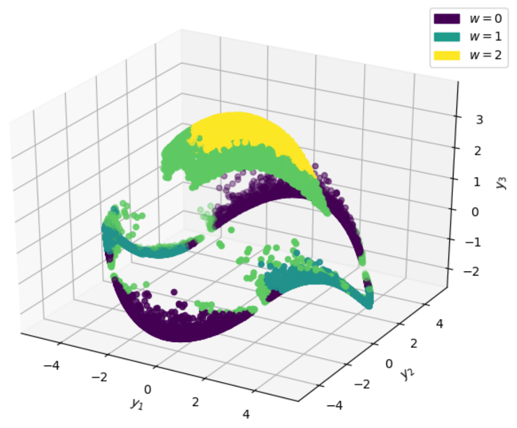
(c)

Figure 3.8: Three snapshots of the projection of the 20000 non-interacting datapoints onto the first three principal components. The phase 2 points are clearly separated from the two other phases. Points belonging to phase 0 and 1 show a different behaviour but are still not completely separated.

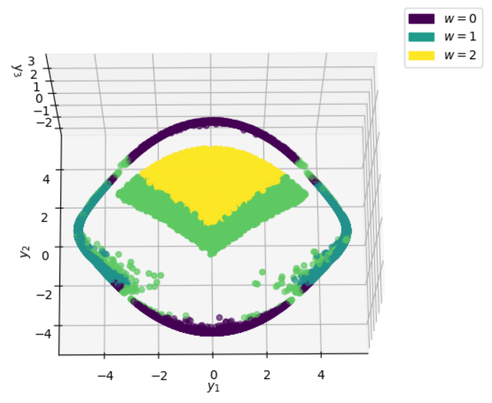
see in the next section.

All in all, with the data we are working we can expect to see a clear separation of it even when only extracting a few components.

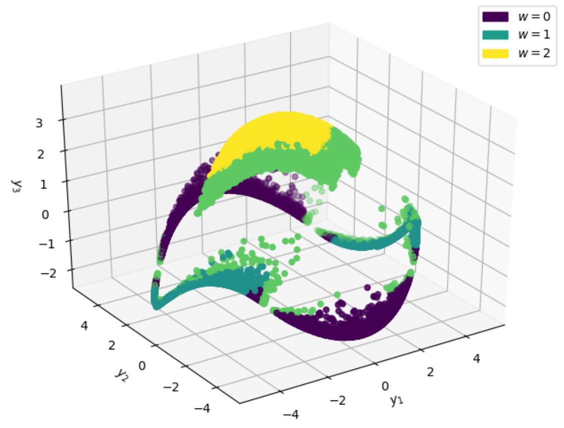
Having said that we want to obtain more insight into these representations on the data. The presence of a triangular form for the phase 2 points suggests that the phase 0 and 1 points should also be divided so explicitely. The fact that they are not separated that much means that they are more similar to each other then the phase 2 points are to each of them. To show that we would require more then three dimensions. Nonetheless we can show that the contact points of phase 0 and phase



(a)



(b)



(c)

Figure 3.9: Three snapshots of the projection of the 20000 non-interacting data-points onto the first three principal components with the critical points labelled in light green color. The graphs are the same as in picture 3.8

1 projections are indeed on the critical lines on the phase diagram.
To do that we consider that the equations of the critical lines are $\lambda_1 = 1 - \mu$, $\lambda_1 = \mu - 1$ and $\mu = -1$ and we assign the same label to all the data points at distance of $\epsilon = \pm 0.5$ from these lines. In this way we can see which points in the PCA projection are close to the critical lines. The results for this are shown in Figure 3.9. Here, the points close to the transition lines are colored in green.
From these new images it emerges that the points close to the critical lines are indeed at the boarder of the different clusters of the phases. In particular, they are more clearly defined for the phase 2 and less defined for the other two phases. Yet, we can see that where the phase 0 and phase 1 connect we have the phase transition points.

By using the PCA we reconstructed the phase diagram of the model. As already mentioned a real definition of the division between phases 0 and 1 would require to exploit the information on the other directions (eigenvectors) of the decomposition.

3.2.2 Interacting Data

We now turn the attention to study the interacting Kitaev model (Eq. (1.133)) dimensional reduction. We perform PCA on 1223 samples of the correlation functions $C(k)$ and $F(k)$, 710 of which calculated in the trivial phase with winding number 0 and 513 in the supercoducting topological phase at winding number 1. Thus the design matrix has shape (1223×200). The 200 eigenvalues of the decomposition are plotted in Figure 3.10. The plot shows that the eigenvalues decrease slowly

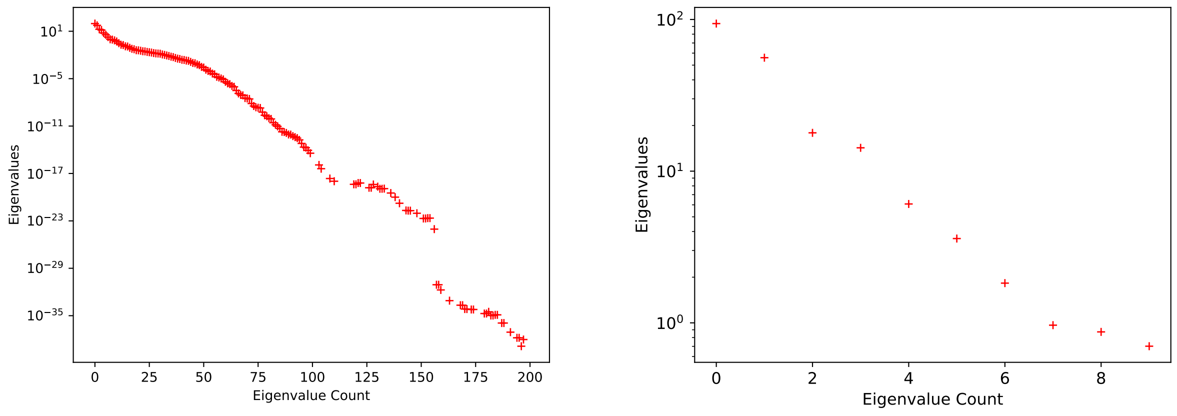


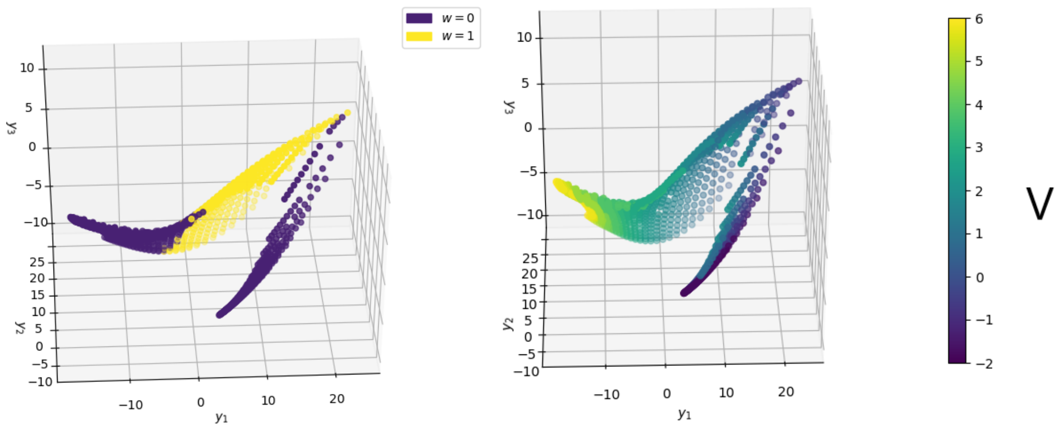
Figure 3.10: Plot of the eigenvalues of the correlation matrix for the interacting model. We can compare the behaviour of the data to the eigenvalues of the non-interacting case and see that they do not go to zero as fast.

compared to the data of the non-interacting case. That means that less information is encoded in the first principal components. For us this means that either we can expect less informative plots of the projections on the first three components and that it is going to be hard to separate the phases of this model.

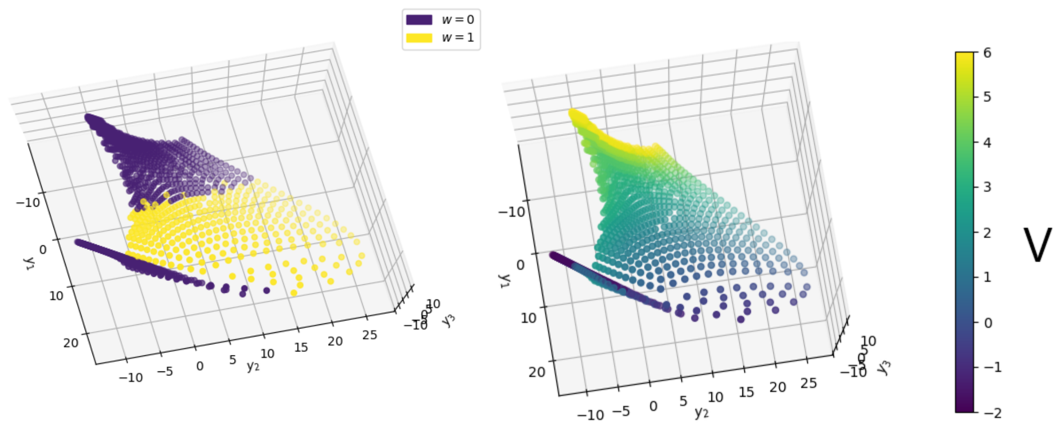
Quantitatively, the less correlation in the data is expressed by lower values of the explained variance of the first components. The first two eigenvalues have an explained variance of 71.1%, which is comparable to the 75.7% of the non-interacting case, but the explained variance of the first three components is 84.7% against the 92.5%.

The complexity of the data is evident in the 3D plots of the projections on the

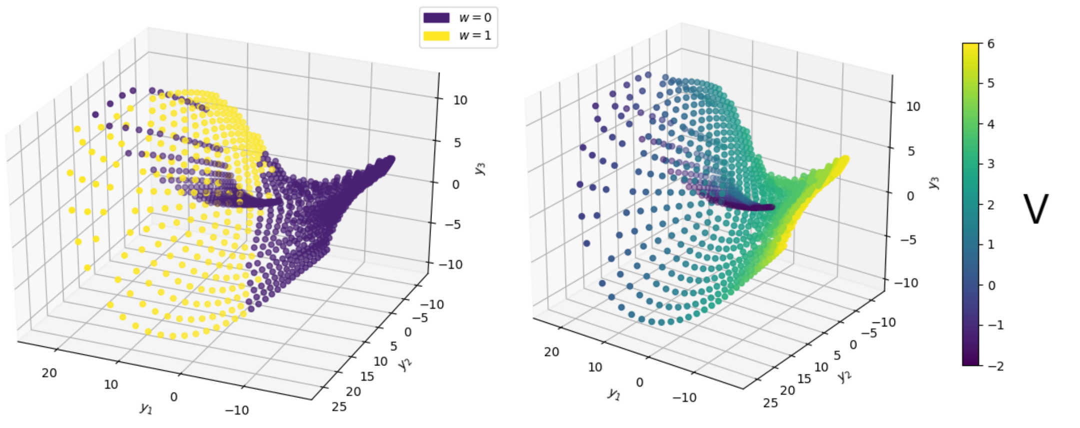
first three components of the interacting data. Those are shown in Figure 3.11 as 3D plots of the same data under different points of view. The column on the left of Figure 3.11 shows different points of view of the same data labelled with their topological phase (0 and 1). Column on the right shows exactly the same data and the same points of view of the graphs on their left but every datapoint is labelled with the potential it was calculated at. Thus, for example, one can check from 3.11c that the points with potential $V = 6$ (right panel) are in phase 0 (left panel).
Interpreting these plots is much more challenging than the non-interacting ones. Nonetheless, there are some behaviours of the data that can be understood. First of all from Figure 3.11a we can see that there is a first neat separation of the data in phase 0 and 1 (left graph of 3.11a) when the potential of interaction grows from low negative to the first positive values (right panel of 3.11a). The data seems to grow along the third principal component y_3 and then change phase at the peak, while preserving the phase 1 label going down.
Cross referencing this details with the view from above of Figure 3.11b (left panel) we can see the exact point where there is a phase transition. Contrarily to the phase diagram of the interacting model, Figure 3.3, in which points along a fixed value of the potential can be in a different phase, here we see that the data points align in the directions of the same phase. Consequently, the superconducting sector of the phase diagram, which changes in a diagonal fashion for increasing V, is recreated slightly rotated by the PCA.
The superconducting phase points (yellow points in the left column panels) are very spread out in their projection on the principal components. This is true in particular if compared to the dense distribution of the trivial phase points (for high V) which are spanned over many more values of V but still so concentrated.
The two sets of blue points correspond to the trivial phase sector and the (Incommensurate) Density Wave sector and are separated by the yellow, superconducting phase. Even though they are in the same topologically trivial phase they are not clustered together. This once again tells us that they are not similar data and thus will not be easy to classify. In fact, we will see that only a network able to recognize their general similarities will be able to see them as in the same phase.
Having said that, it would be hard to notice these behaviours if one were to do the PCA on this data without labels. Their projections would create a homogeneous manifold which would not help to cluster the points but, considering the results of the PCA o non-interacting data, give hints on the shape of the phase diagram.



(a) Lateral, rising, view



(b) Top, peak, view



(c) Lateral, decreasing, view

Figure 3.11: Three snapshots of the projection of the 1223 interacting datapoints onto the first three principal components. Each of the snapshots is presented with the labels of the phase and the interacting potentials to help distinguish the two phases. If we consider the projections of the data as moving along the third principal component, then the data are taken respectively at the rising part, the peak and then the decreasing part. The colorbar shows the values of the potential.

3.3 DNN to Predict the Phases of the Interacting Model

The PCA decomposition of our data turns out to be very useful to understand how to work on our data and what to expect. On one hand, we expect the non-interacting data to be easier to recognize and classify. On the other end we saw that the line between phase 0 and phase 1 data is "thin" in a sense that it requires more information to be understood (Figure 3.11). We realize then that the interacting data is more intricate then expected, meaning that the phase separation lines are not easy to detect as in the non-interacting case.
The information we just gathered seems very general and not too informative. Instead, we can use it to find the most fitted algorithm to work on and to understand while it might fail in its generalization task.
In this part of the results we show how we worked on a Fully Connected Neural Network for classifying the topological phases of the interacting model. Our purpose is to train a network with the non-interacting data and apply the same network to interacting data to understand if there is any feature that was learned by the DNN and to see if these features are enough to classify data generated by a different, interacting, Hamiltonian. Moreover, our task has a more general point of view. We are trying to show if it possible to make a Deep Learning algorithm for learning a model of which we know the phase diagram and predict the phase diagram of a related model of which we have less knowledge. The task we require our algorithm to perform is not a generalization on the training data to the test data of the same distribution. We are asking it to generalize on test data of a different model. Therefore, we do not expect it to behave regularly and we will not focus on the performance of the network on its own data rather than its performance on the new data.
Before diving into the results, it is important to state the decision process that led us to choose a Deep Learning algorithm rather than a Machine Learning one. The first reason has just been mentioned, that is we expect the generalization we are asking for to be out of the scope of a ML algorithm. Because in order to be successful the algorithm will have to understand deep structures of the data. The second reason is that we want to investigate which features of the data are learned and why they are important in the decision process of the algorithm. Neural Networks, even though their *black box* nature does not allow it in principle, can indeed be "opened up" to

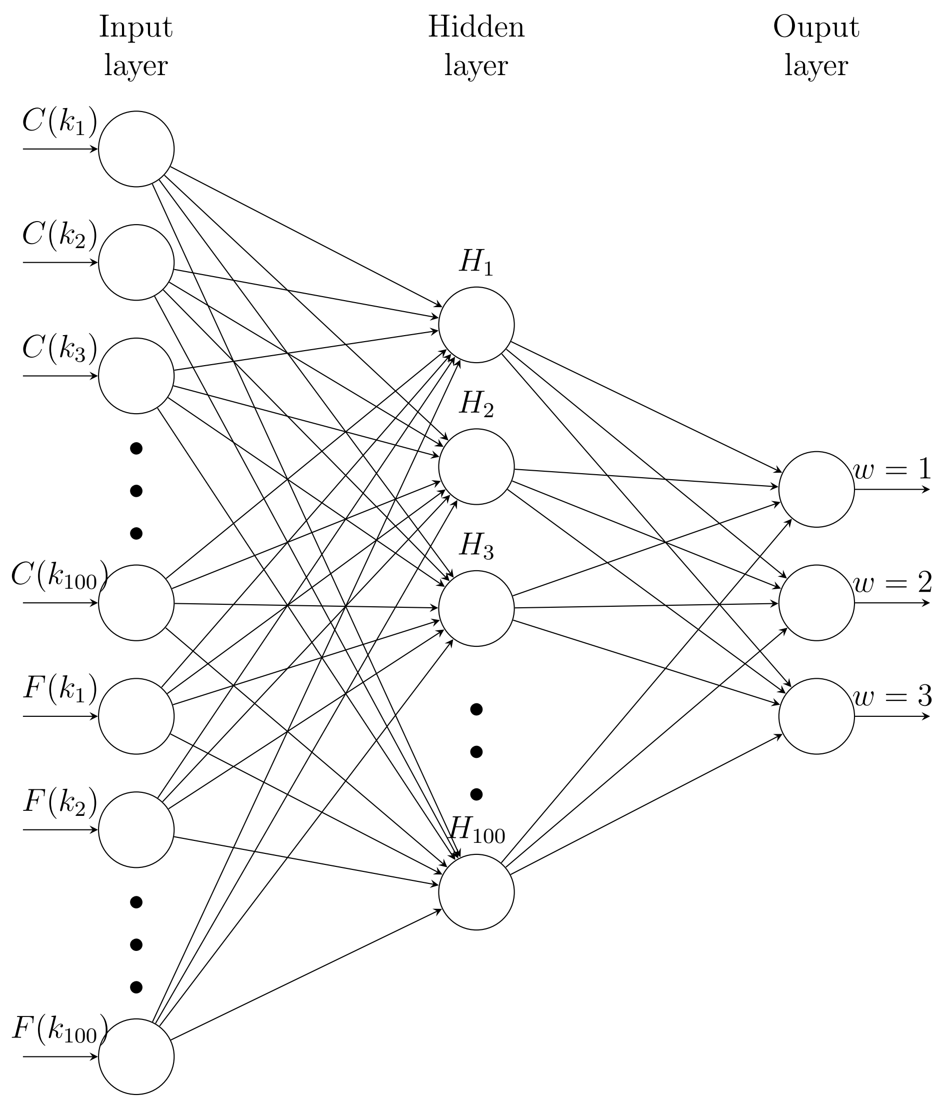


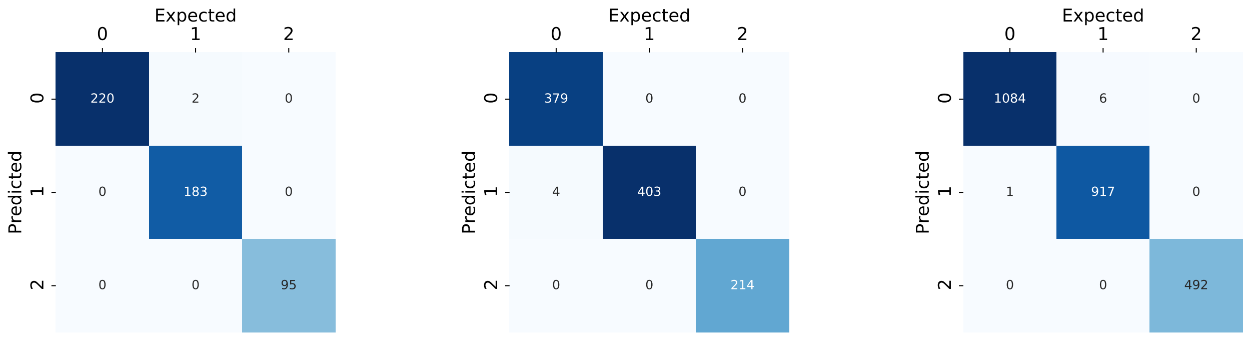
Figure 3.12: Architecture of the Neural Network used for the results of this chapter. The inputs are the 200 values of the correlation functions $C(k_i)$ and $F(k_i)$ with $i = 1, \ldots, 100$. The hidden layer has 100 neurons and then the output has 3 neurons to classify the topological phase of the input with a certain probability.

understand which features were learnt (as it is done in many papers on DL applied to Condensed Matter Physics such as [14]). Lastly, the PCA already showed that data is not clustered as expected and thus can be difficult to analyze and interpret.

3.3.1 Training and Testing on Non-Interacting Data

We start by investigating the ability of a simple Neural Network to predict the topological phases of the non-interacting data.

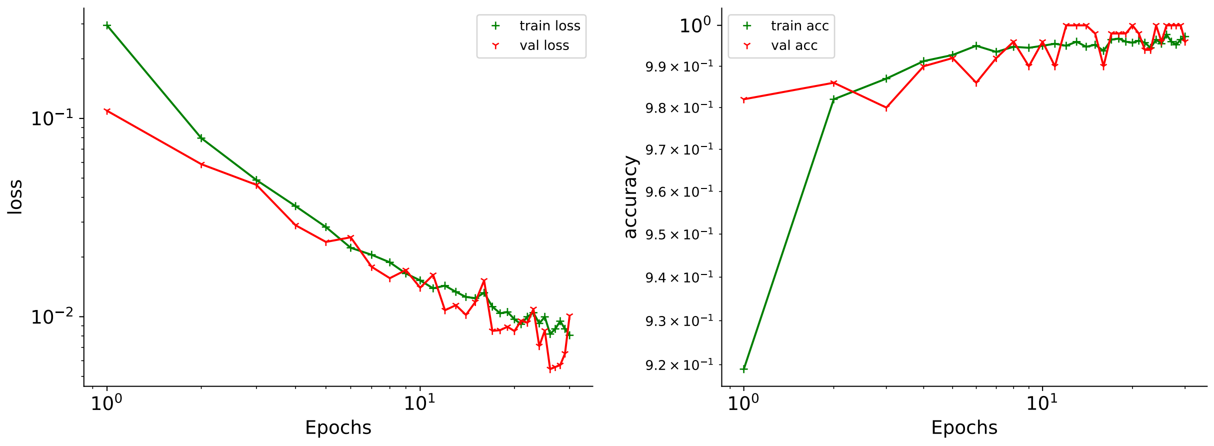
For this purpose we devise a basic Neural Network which has 200 input neurons corresponding to the values of the correlation functions $C(k)$ and $F(k)$, one hidden layer of 50 neurons and an output layer of 3 neurons corresponding to the probability of the input to belong to either one of the three topological phases. We train the NN with the ADAM algorithm, using categorical cross entropy validation and measuring the accuracy of the algorithm by simply counting the number of correctly classified points. We did not focus on the choice of hypeparameters at first, see discussion below. The activation of the first hidden layer is the sigmoid function, for the output layer the SoftMax. We train the network with batches of 10 samples over 30 epochs using the three datasets described at the beginning of the chapter. The results of this first test are shown in the confusion matrices in Figure 3.13. The accuracies of



(a) $\mathcal{D}_1$ (b) $\mathcal{D}_2$ (c) $\mathcal{D}_3$

Figure 3.13: Confusion matrices of a Neural Network trained with batch size 10 for 30 epochs for the different datasets $\mathcal{D}_i$. The confusion matrices show the predicted (columns) vs expected (rows) classifications of the data. For example the NN trained on $\mathcal{D}_1$ misclassified 2 points belonging to phase 0 (first raw) as points belonging to phase 1 (second raw).

the NN trained with the different datasets are for $99,6\%$ for $\mathcal{D}_1$, $99,9\%$ for $\mathcal{D}_2$ and $99,6\%$ for $\mathcal{D}_3$ on the test set for the data shown in Figure 3.13. During some first runs on the trainings the accuracy even reached 100% for some models. In addition to the confusion matrices we plot the accuracy and the loss on both the training and validation set of the $\mathcal{D}_1$ dataset only. The behaviour of both quantities is expected: the loss function approaches lower values with the epochs and the validation follows it. This is the sign that the algorithm does not experience neither overfitting nor underfitting. The same happens for the accuracy which converge to the maximum value possible. In this case, as in some others we will see, the validation loss seems



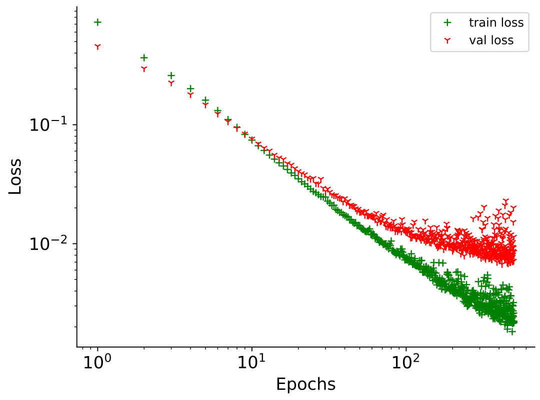
(a) Training Loss (b) Training Accuracy

Figure 3.14: Training Loss and accuracy on both the training and validation set for 30 epochs of the $\mathcal{D}_1$ dataset

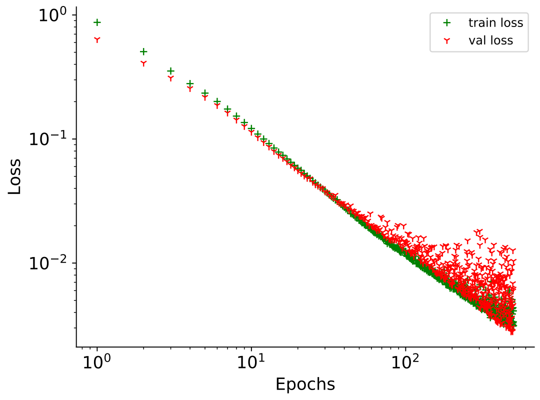
to be lower then the training loss. Although it is in theory not possible to perform better on unknown data, such as the validation ones, this is still possible when the validation data is particularly "easy" to classify (less noisy, *e.g.*).
The lower loss values of the validation set can also depend on the fact that the loss of one epoch of training is an average of how the algorithm performs from the beginning of the epoch-when it could classify poorly- to the end. The loss on the validation, instead, is measured after a whole epoch so the algorithm is indeed slightly more "experienced" then during the previous epoch of training.
Having said that, a large and constant difference in training and validation cannot be accepted as a good "fit" of the data. It is actually considered overfitting. In order to subvert this issue we retrained the network every time there was a large, negative, gap between validation and loss.
Moreover the behaviour of both the training loss and accuracy has to be controlled more specifically, we will see soon what we mean by that, for now we just wanted to show the main parameters introduced in Chapter 2.
In general, from this data we understand that even a simple neural network classifier can identify the phases of the non-interacting model with great precision. This is something we somehow expected when analyzing the principal components.

3.3.2 Testing on Interacting Data

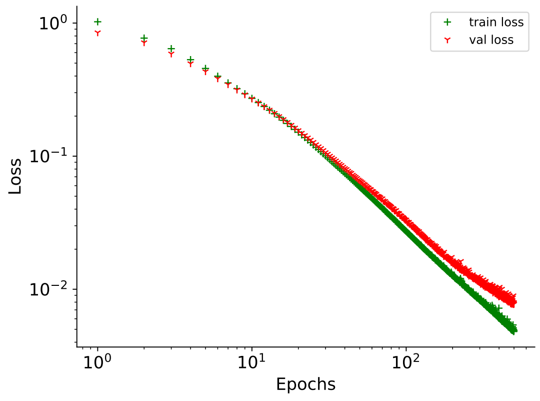
With this positive results we apply the three models trained with different $\mathcal{D}_i$ to the interacting data.
The classificators perfectly recognize the phases of the interacting model for $V = -2, -0.2, 0.2, 0.8$ with a 100% precision. We decided to test on these values because they are both far from the phase transition lines ($V = -2$) and localized around them (all the other values).
For potential $V = 6$ the three neural networks are not able to assign even one correct phase to the data points. All the points with winding number 0 are classified as having winding number 1. This problem was in a certain way predicted by looking at the more complicated shapes of the correlation functions calculated at this high potential.
To make sure the problem of classification in the interacting case is not due to a poor performance of the DNN we try switching a few hyperparameters. We changed learning rate, the number of layers in the DNN, the number of neurons per layer, the type of activation from sigmoid to ReLu and eventually the batch size and the epochs. None of this changes seem to produce any relevant different in the classification of the interacting data at $V = 6$ except from the batch size.
To be more precise we analyzed the behaviours of the loss functions for different batch sizes and epochs. We noticed that the loss function (for both training and validation) goes to zero with a power law only for large batch sizes such as the ones plotted in Figure 3.15. The plots are all generated after the same number of epochs, 500. The difference is in the batch size which ranges from 100 to 1000. We can see that the two plots with lower batch size are more likely to suffer overfitting. On the other hand, the graphs with batch 500 and 1000 go to zero exponentially and are followed by the relative validation error.
We need to emphasize that this analysis would not be necessary if we only dealt with the non-interacting value, since our confusion matrices are almost perfect. This analysis is important to recognise what elements we should control when asking for a different generalization such as predicting the phase of the interacting model.
We interpret the precision of the batch 500 and 1000 trainings as a sign of possible quality of generalization on interacting data. In fact training the neural network with batch 500 does indeed make the predictions on $V = 6$ better. Even though the classifier still has an accuracy of 0% on this data, the output probabilities of the neurons are different. That is, the probability of the wronly assigned phase



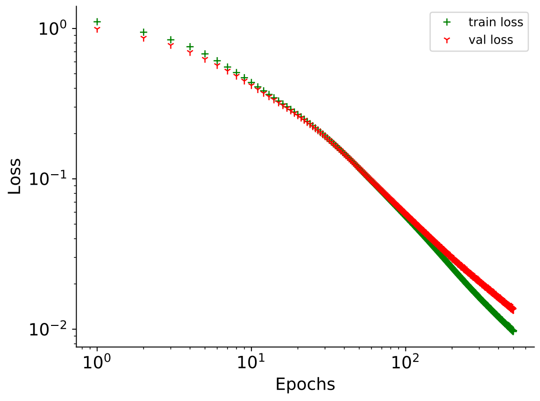
(a) Batch = 100



(b) Batch = 200



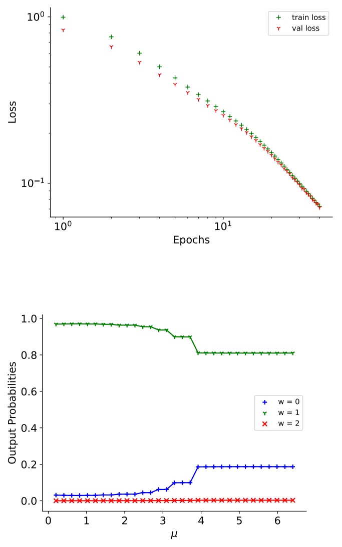
(c) Batch = 500



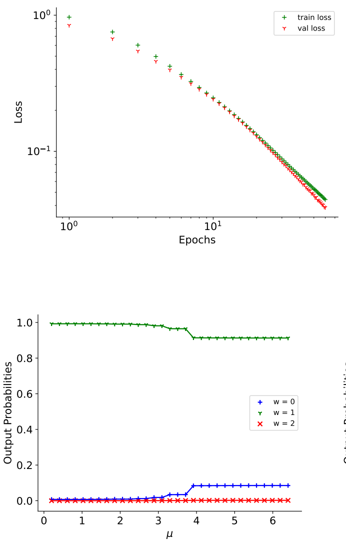
(d) Batch = 1000

Figure 3.15: Training and validation losses of the NN trained on the $\mathcal{D}_1$ dataset. All the plots are generated for 500 epochs of the training.

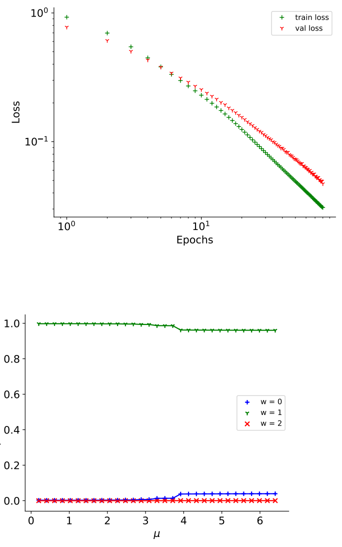
starts decreasing as we approach the most fitted batch size for the model, as shown in Figures 3.16. The plot of the probabilities of the output neuron is the different values from 0 to 1 that everyone of the three output neurons assign to a datpoint to belong to phase 0, 1 or 2. As expected the phase 2 neuron outputs zero values for all the datapoints of $V = 6$. As we can see, although the classification is still incorrect because the network classify the points as belonging to the superconducting phase, there is a slight improvement for the network trained with 40 epochs as compared to 60 or 80. On the same level, the plots of the training and validation loss seem to converge better than the others. This concludes our first consideration on the role of the batch size and on the generalization capabilities.



(a) Epochs $= 40$



(b) Epochs $= 60$



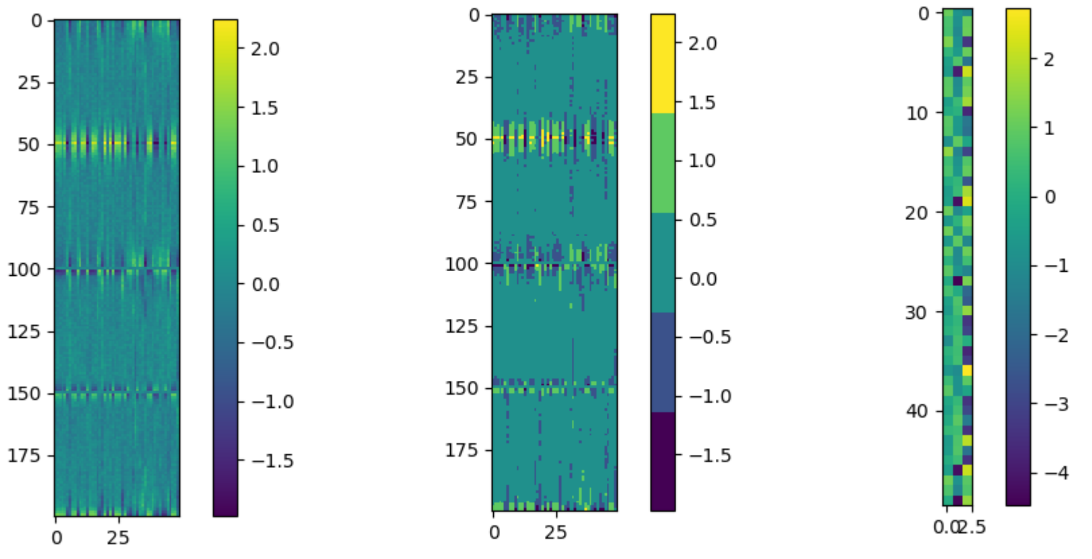
(c) Epochs $= 80$

Figure 3.16: Training and validation losses of the NN trained on the $\mathcal{D}_1$ dataset and the corresponding output probabilities of the neurons for data points at $V = 6$.

3.3.3 Opening up the Neural Network

Since our neural network seems to fail in the classification task we proposed we try to analyze what could be the cause of mistake by understanding what the network learns. This is typically done by investigating the weights of the network and the activations of the layers.

Weights of the NN The first layer of the network is made up of 50 neurons. The input is instead made of 200 values of the $C(k)$ and $F(k)$. Therefore, the weight matrix connecting the input to the first layer has dimensions 200×50, while the weight matrix from the first layer to the output layer is 50×3. The plots are showed in Figure 3.17, to make it more readable the weights are plotted both in their original form and in a discretized color version to erase the noise of small values. From both Figure 3.17a and 3.17b we can see that the weights are respectively large around the mulitples of 50 input neurons. These weights are multiplied with the extremal values and the central values of the correlation functions, namely their values around $k = 0, \pi, 2\pi$ for both $C(k)$ and $F(k)$. In particular from picture 3.17b it is clear that most part of these curves is obscured by weights around 0 that cancel each other out, while the values around $k = \pi$ of the correlation function $C(k)$ seem to be very relevant at this step of the neural network "forward pass" process.

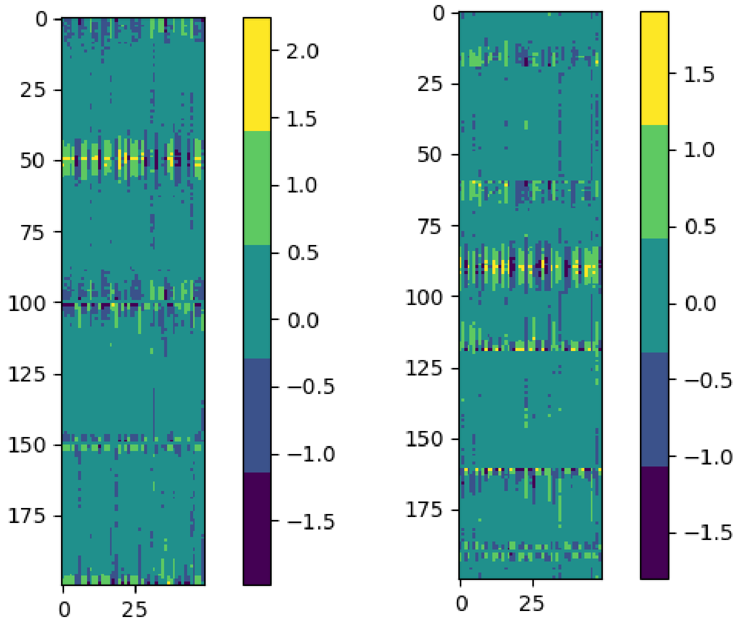


(a) First Layer (b) First Layer (c) Output Layer

Figure 3.17: Weight matrices of the NN trained with $\mathcal{D}_3$ for 30 epochs and batch 10. The plot on the left is the weights connecting the input layer to the first hidden layer. The one in the middle is plotted with discretized color to show that most weights are around zero. The last bar shows the weights connecting the hidden layer to the output.

Unfortunately, it looks very hard from picture 3.17b to predict the behaviour of the outcome of the network. Thus we check the activations of the layers.
In order to check that the values of the weights grow larger close to $k = 0, \pi, 2\pi$ we trained the same neural network but inverting the order of the input neurons. By that we mean we divided every single sample of the data in 5 sectors $0, 1, 2, 3, 4$, shuffled them into $4, 3, 1, 0, 2$ and fed it to the neural network for training. The result is shown in Figure 3.18 where we plotted the weights of the regular input next to the new weights so that it can be inspected that the learned weights are still larger in the values around $k = 0, \pi, 2\pi$

Layer Activations of Non-Interacting Data The layer activations in the ML language are the outputs of a layer, that is the outcome of each neuron after the input has gone through the activation function.
Considering the neural network trained on dataset $\mathcal{D}_1$ for 30 epochs and batch size

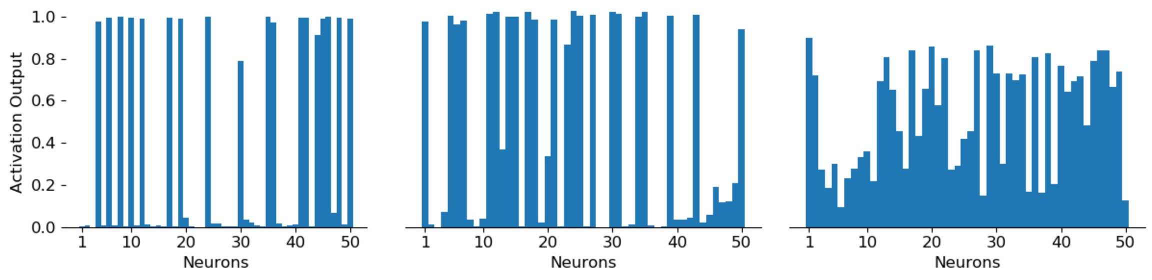


(a) (b)

Figure 3.18: Comparison of the learned weights of the Neural Network trained with dataset $\mathcal{D}_3$ with different inputs. On the left, the model was trained with the usual input. On the right, the model was trained with regular data but the order of its points was shuffled (in the same manner for every data entry) in 5 blocks of 40 neurons each. For example the second block going from neuron 40 to 80 was moved down of one block and it is indeed reproduced by the neuron weights from 80 to 120 in the plot to the right.

of 10 samples we first plot the activations of the first hidden layer when we feed it with the data it was trained on. We find that the behaviour of the activation changes only slightly between points of the same phase.

Figures 2.4 show the activations when the network is fed with three training set values with corresponding to phase 0, 1, and 2. Let us recall that the input data is firstly linearly transformed by the weights (such as the ones in Figure 3.18) and biases of the network. Then it goes through a sigmoid activation function in order to produce the outputs visualized in Figure 3.19. Mathematically, let us write the array $[C(k), F(k)]$ of 200 elements as one input array $[I(k_i)]$ where $i = 1, \ldots, 200$.



(a) $w = 0$ (b) $w = 1$ (c) $w = 2$

Figure 3.19: These plots show the values that the first layer of the network gives as output after it is fed with, respectively, data from phase 0, 1 and 2 of the non-interacting dataset.

This inputs goes through a first linear transformation:

$$Z_m = \sum_{i=1}^{200} W_{im} I(k_i) + b_m, \tag{3.8}$$

and then throguh the sigmoid activation:

$$a_m = \sigma(Z_m) = \frac{1}{1 + e^{-Z_m}} \tag{3.9}$$

which produces numbers a_m in the range [0,1].

Layer Activations of Interacting Data When we feed the neural network with the interacting data we can see from the activations of the first layer what changes in the classification process.
In Figure 3.20 the plot on the left shows the activations of the interacting data at $V = 0.8$ which undergoes the phase transition when μ is between 3.4 and 3.6. Indeed, the behaviour of the activations changes abruptly in that interval. That is why it perfectly recognizes the two phases (recall that none of the models has problem recognizing phases at low values of V). Notice how the activations resemble the behaviour of 3.19b and 3.19a before and after the transition.
Conversely, at the interacting potential $V = 6$ there is no sensitive change in the activations and the active neurons are the same of 3.19b meaning that it recognises this (wrongly) as a phase with winding number 1. Some neurons start activating when μ varies, but only enough to reduce the certainty of the prediction (which goes to a lower 70% for $\mu = 5.6$ for example) even though it is still wrong. To understand

the slight change we can confront with 3.5a and 3.5a where the correlation function $C(k)$ is changing.
These two outputs are from the network trained with 30 epochs and batch size 10. We will use these types of results to compare what is learned by the DNN compared to the CNN.

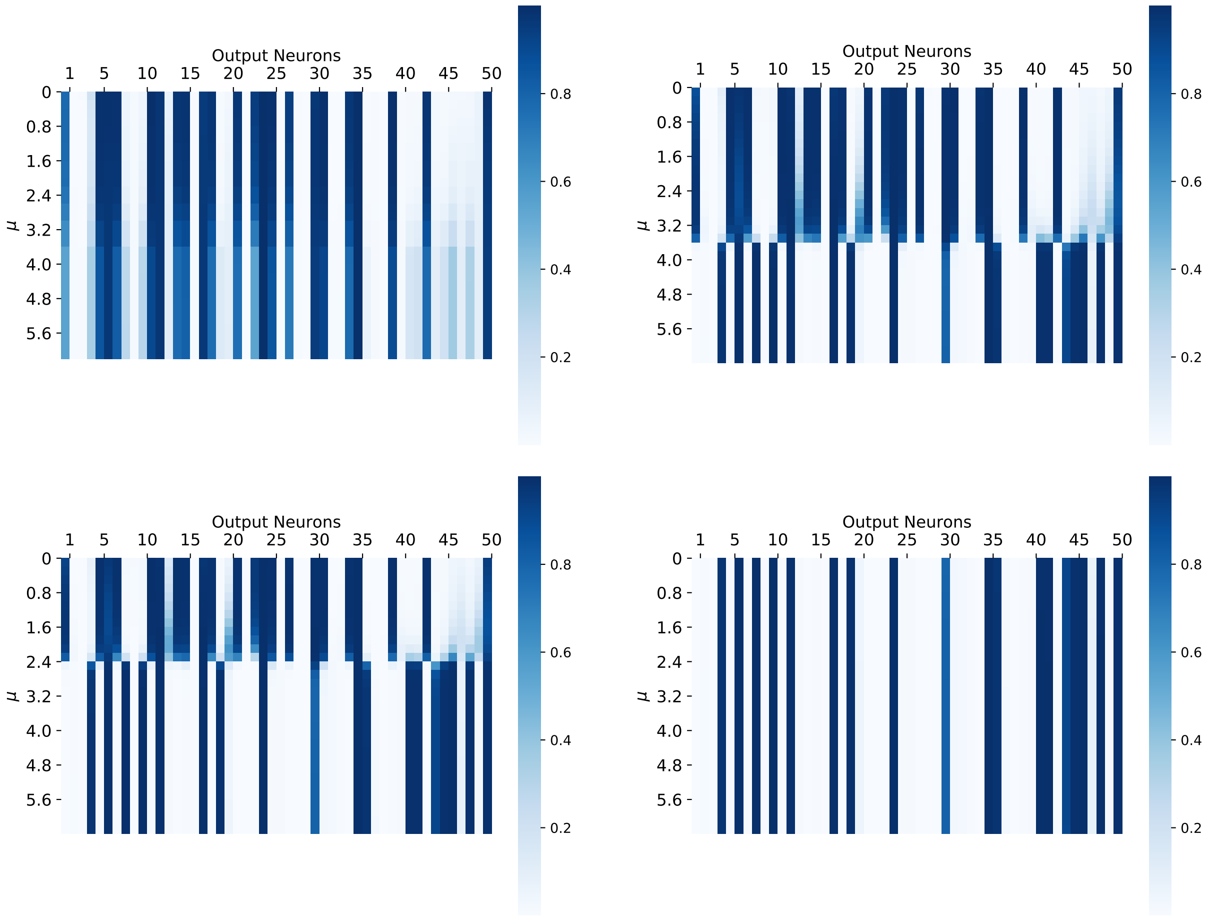


Figure 3.20: Activations of the first hidden layer of the DNN for interacting data. The plots show the activations at $V = 0.8$ and $V = 6$ for all the values of μ. The phase transition happens at $\mu = 3.4, 2.4$ for $V = 0.8, 0.2$ respectively, where the activations change.

3.4 Convolutional Neural Network

The first part of these results shows how and possibly why a feedforward neural network fails to assign the right topological phase to data generated at high potential

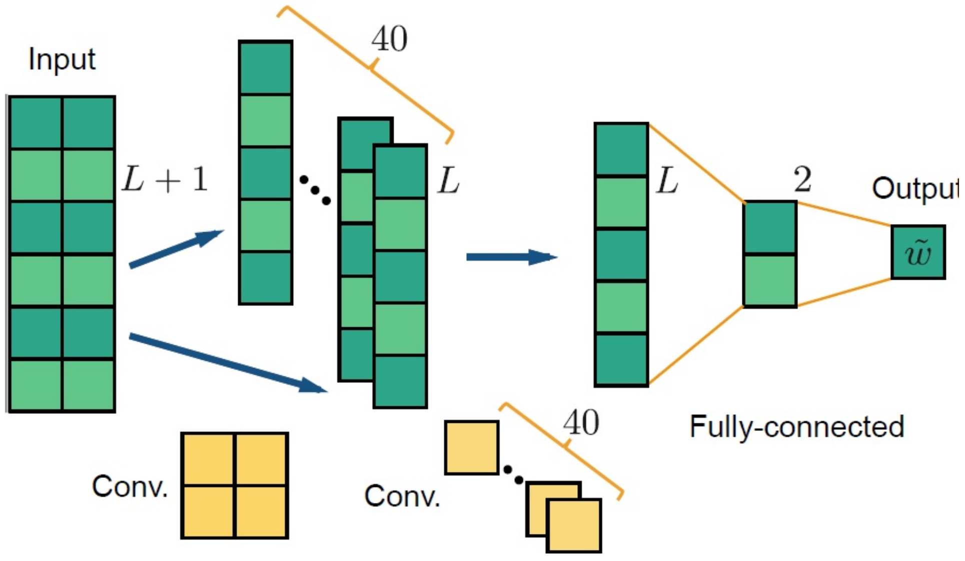


Figure 3.21: Representation of the Convolutional Network used in these results section. The algorithm performs a 2D convolution extracting 40 features. Then, a 1D convolution extracting one feature that is finally put in a DNN for classification. Picture taken from [14].

V in the interacting Kitaev chain.
One of the reasons that a DNN cannot correctly classify our data is that it does not learn the spatial features of it and does not take it into account. This is why we decided to apply a Convolutional Neural Network. We were also inspired by the job of Zhang *et al.* [14] who applied CNN's to classify the topological phases of generic models in the AIII symmetry class. The data they used was a series of values of the Hamiltonian at different values of momentum k.
In our work we present an algorithm realized in a simile manner to that of [14] but using correlation functions as data.

3.4.1 Structure of the Network

A scheme of the Convolutional Neural Network (or ConvNet) is shown in Figure 3.21 The input is a matrix 2×100 with the structure factors $C(k)$ and $F(k)$ as columns. The first convolution uses $M = 40$ different kernels of dimension (2,2) to extract 40 features called receptive fields which are 1×99 arrays:

$$\tilde{Z}_i^k = W_{1,1}^k C(k_i) + W_{1,2}^k F(k_i) + W_{2,1}^k C(k_{i+1}) + W_{2,2}^k F(k_{i+1}) + W_0^k \tag{3.10}$$

where $k = 1, \dots, 40$ labels the M receptive fields, W_0^k is the bias and $i = 1, \dots, 99$ because the convolution stops at $L - 1 = 99$. $\tilde{Z}_i^k$ is a tensor representing the 99 components of the 40 extracted features. These are passed through ReLU activation, that is

$$Z_i^k = \max\{0, \tilde{Z}_i^k\}. \tag{3.11}$$

The job of this non linear activation is simply to not consider the activations with negative values. Then, a 1D convolution (of kernels which are obviously (1,1)), produces one single receptive field of 99 elements:

$$\tilde{A}_i = \sum_{k=1}^{M} w_k Z_i^k + w_0 \tag{3.12}$$

$$A_i = \max\{0, \tilde{A}_i\} \tag{3.13}$$

This result is in just a simple linear combination of the previous 40 activations. The convolution steps produce an array which is now fed to a feedforward neural network with a hidden layer of 5 neurons. Therefore the array A_i is transformed by a matrix 99×5:

$$\tilde{F}_\eta = \sum_{n=1}^{L-1} S_{\eta n} A_n + B_n, \tag{3.14}$$

$$F_\eta = \max\{0, \tilde{F}_\eta\} \tag{3.15}$$

and then transformed by a 5×3 matrix

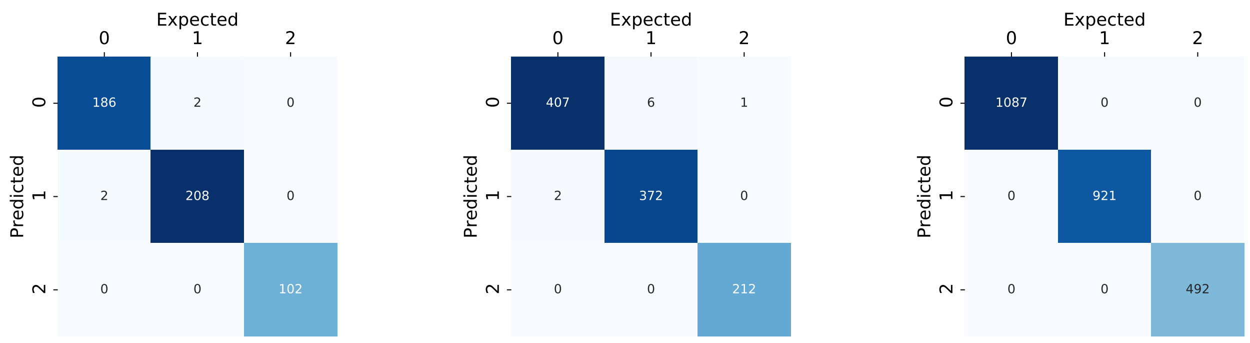
$$O_l = \sum_{\eta=1}^{5} P_{l\eta} F_\eta \tag{3.16}$$

$$\tilde{\omega}_l = \frac{e^{O_l}}{\sum_{l=1}^{3} e^{O_l}} \tag{3.17}$$

where the last nonlinear transformation is the SoftMax activation and $\tilde{\omega}_l$ is the probability that each of the three outputs gives to a datapoint to belong to one of the three phases. All in all, the matrices and arrays $W_{i,j}^k, W_0^k, w_k, w_0, S_{\eta,n}, B_n, P_{l,\eta}$ are the parameters to be learnt.

3.4.2 Reproducing the Interacting Phase Diagram

Testing on Non-Interacting Data We investigate the power of generalization of this model. From what we gathered in the last section it is best to work with large batch sizes. In the training procedure we get optimal results on the non-interacting test set, as it was for the DNN. The results are shown in the confusion matrices



(a) $\mathcal{D}_1$ (b) $\mathcal{D}_2$ (c) $\mathcal{D}_3$

Figure 3.22: Confusion matrices of the ConvNet. The CNN classifies the non-interacting data with accuracy of (a) 99.2% for the $\mathcal{D}_1$ dataset, (b) 99.1% for the $\mathcal{D}_2$ dataset and (c) 100% for the $\mathcal{D}_3$ dataset.

in Figure 3.22. The CNN has the same precision of classification of non-interacting data as the DNN (values in the caption of Figure 3.22).
In addition to the confusion matrices we plot the loss and accuracy of the model trained with the $\mathcal{D}_2$ dataset. As already mentioned this is common procedure to prevent underfitting (both validation and training loss does not converge to 0) and overfitting (trainig loss convergences but validation loss does not).

Testing on Interacting Data We are interested in the results on the interacting data. Surprisingly, the CNN perfectly classifies them. The plots 3.24 show the predictions of the convolutional neural network. This shows the output of the three neurons of the network, for different values of μ for fixed values of the potential V. The predictions for $V = -2, 0.2, 0.8$ are less interesting because we already showed by the data PCA and the DNN implementation that they are easy to recognize. Nonetheless, it is interesting to notice the sharp drop of the neurons probability prediction of the values at $V = 0.2, 0.8$ where the phase transition occurs. The $V = 6$, instead, represents the Charge Density Wave sector with a trivial topological

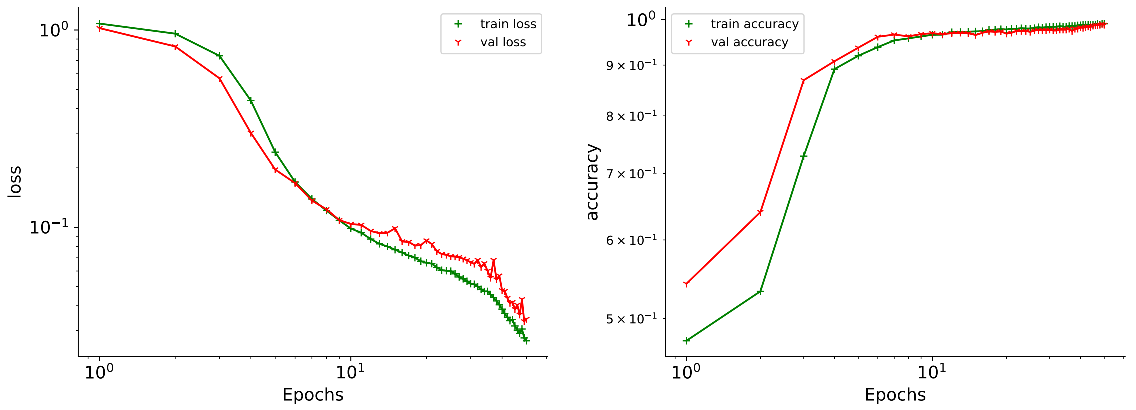
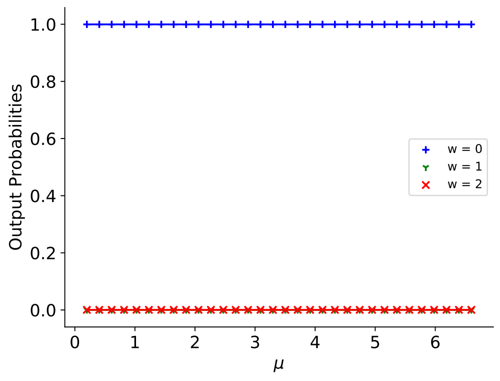
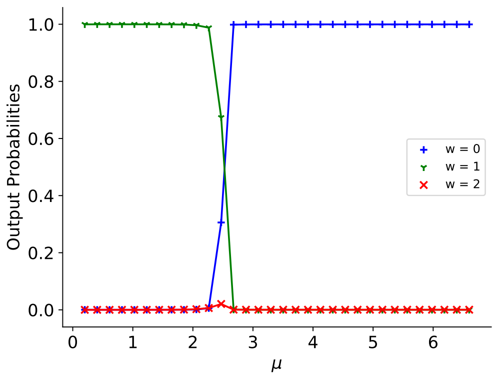


Figure 3.23: Loss and Accuracy of both the training and validation sets for 50 epochs of the $\mathcal{D}_2$ dataset.

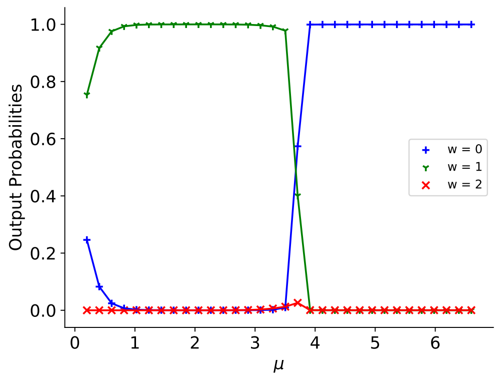
phase that is much more difficult to find.
Since these results are promising we want to see if the same happens for datasets $\mathcal{D}_2$ and $\mathcal{D}_3$. We find that for batch size 300 also the CNN trained on the second dataset reach a 100% precision on the test set at $V = 6$ (unless diversely specified we will assume 100% accuracy also on the other test data). The same happens for the CNN trained on the $\mathcal{D}_3$ dataset at batch size 800. Therefore, the batch size has to increase with the data in order to have good generalization power on the interacting data.
To validate our results we check the convergence of the loss functions of the models for many epochs of training. Their convergence is shown in the first row of Figure 3.25. By convergence we mean that both the training and the validation loss tend to zero as the number of epochs increases. In addition to that, we check that the loss for the three models suffers from slight overfitting. In fact in the second row of Figure 3.25 we see that in the three models-trained respectively for 30, 50 and 50 epochs at batches 100, 300 and 800- the validation loss starts moving away from the training loss.
It is important, though, that this does not affect the classification for low V, it actually just enhance the prediction abilities for high potentials. That might mean that in order to predict the complicated phases at V=6 the ConvNets need a bit of overfitting to allow for generalization in the sense that we are asking. In summary, CNN's turn out to be much more reliable to understand the data we provided and in fact can be used to predict the whole phase diagram of the interacting model. Figure 3.26 is the reproduction of the phase diagram. For every data entry at different μ and V we plotted the certainty that the CNN feels in assigning topological phase 0



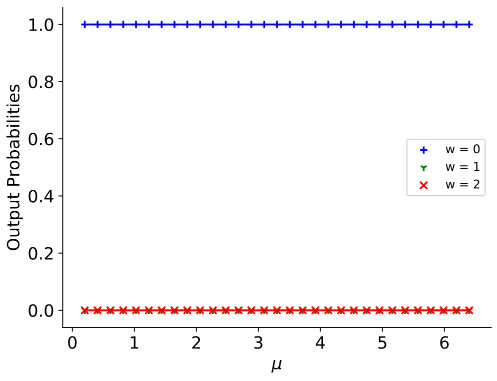
(a) $V = -2$



(b) $V = 0.2$



(c) $V = 0.8$



(d) $V = 6$

Figure 3.24: Some output probabilities of the CNN trained on data at $V = 0$ and tested on data at $V = -2, 0.2, 0.8, 6$, the most interesting values. Conversely to the DNN, the ConvNet perfectly recognizes the trivial character of $C(k)$ and $F(k)$ even at high potential.

or 1 to the point. Comparing it to Figure 3.3 we can affirm it is an accurate result. In the reconstructed phase diagram we even see the points on the critical lines, which have transitioning colors from yellow to blue because the network is not certain how to classify these points. On the other hand, many data points with non zero values of the density of edge modes (on the phase diagram produced with the DMRG) are here labelled with 100% certainty. That is probably due to the fact that the network is asked to produced a "hard" classification (either one phase or the other) for these points with the highest probability possible and that does not leave room for interpretability.

Considering what has been said, it is still very impressive that we were able to

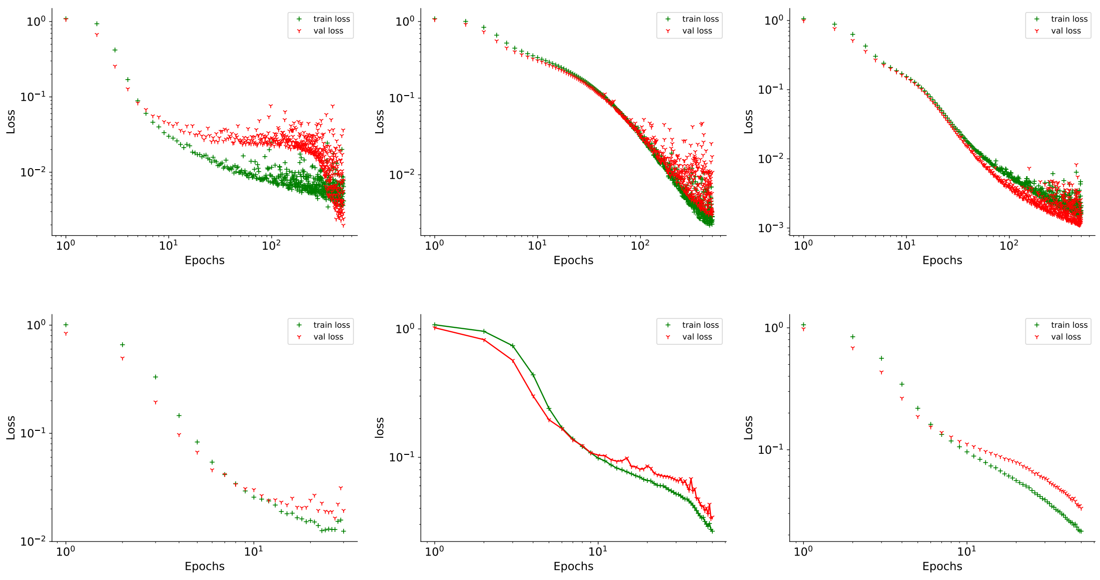


Figure 3.25: Training and validation loss for the models trained on datasets $\mathcal{D}_i$. The top row is the result of 500 epochs of training, plotted in order to check if in the limiting case of infinite epochs the algorithm converges monotonically to zero loss. Bottom row is for the models trained on 30, 50 and 50 epochs respectively when they perfectly classify the points of $V = 6$.

reproduce this diagram, shown in Figure 3.26, with the simple knowledge of the non-interacting model and the phase transitions.

3.4.3 Opening Up the ConvNet

As already mentioned, we want to show what the CNN learns, so we show the activations of the layer of the ConvNet. The activations are the receptive fields. That is, every kernel produces a convolution which results in a single array, this is passed through ReLu activation and that is the activation for that filter. As Convolutional Neural Networks are designed to work mostly on image it is easier to understand what features of the data are learnt and which are discarded.

Non-interacting Data The first convolution of the network is usually supposed to recreate the data or at least some general features of it. In Figure 3.27 we see how the set of 40 arrays of 99 neurons activates after convolution and ReLU activation. We present them as a matrix where each column is a different activation,

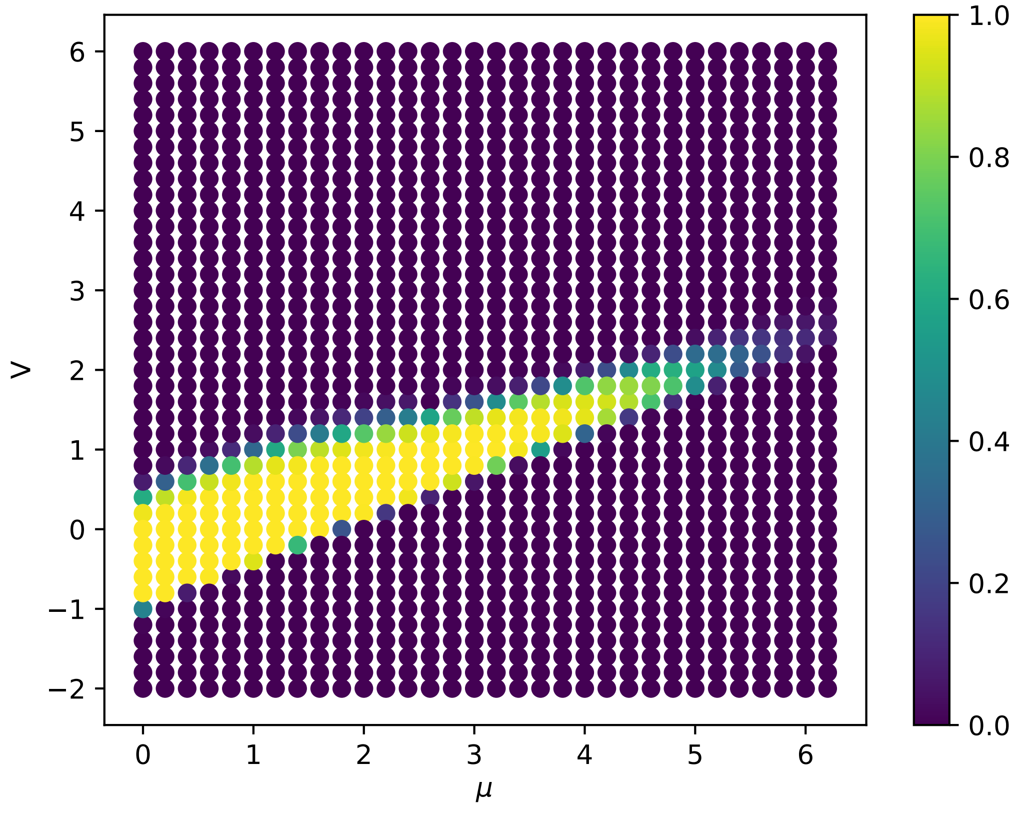


Figure 3.26: Predicted phase diagram: plot of the output of the CNN trained on the $\mathcal{D}_3$ dataset with batch 800 for 50 epochs. In the graph we show the probability of a datapoint to belong to the superconducting phase of winding number 1. Remarkably, the likelihood of a point to belong to a class reproduces the interacting model phase diagram.

or receptive field. Due to the particular form of the ReLu function all the negative values are set to zero. That is a measure to avoid redundance: the idea of CNN is that the negative values would just learn the same features of the positive ones and there is no need of them. In the figure we see that some of the arrays are indeed zero, thus they carry no information.
Comparing the activations for different data points at $w = 0, 1, 2$ we see that the activations for the datapoint at winding 0 are mostly flat (Figure 3.27a), meaning that each receptive field is an array of constant values. The activations of the datapoints of phase 1 and 2 (Figures 3.27b, 3.27c) have a more complicated structure, resembling the wave structure of the input correlations for which they were gener-

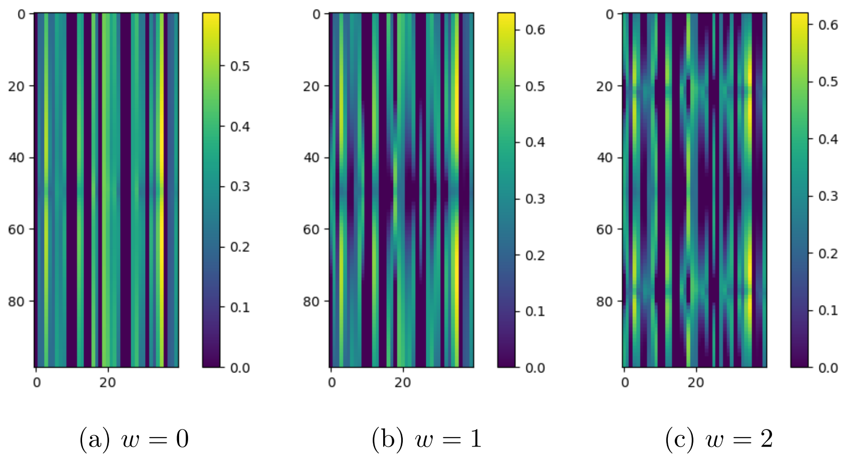
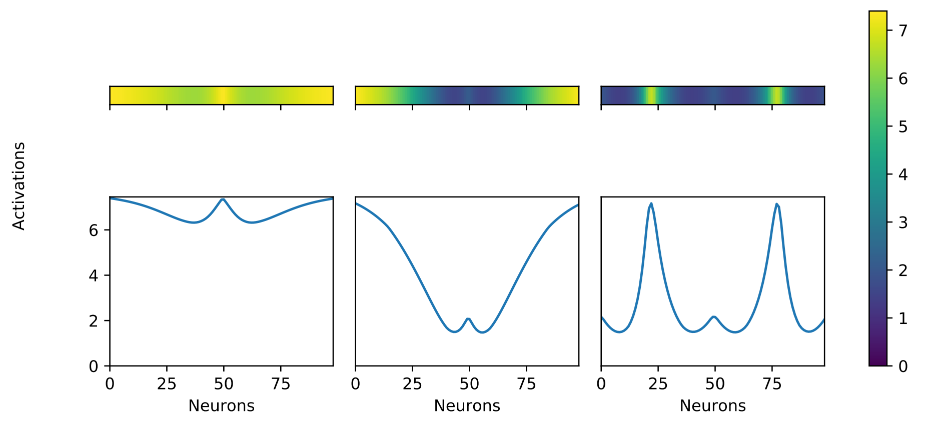
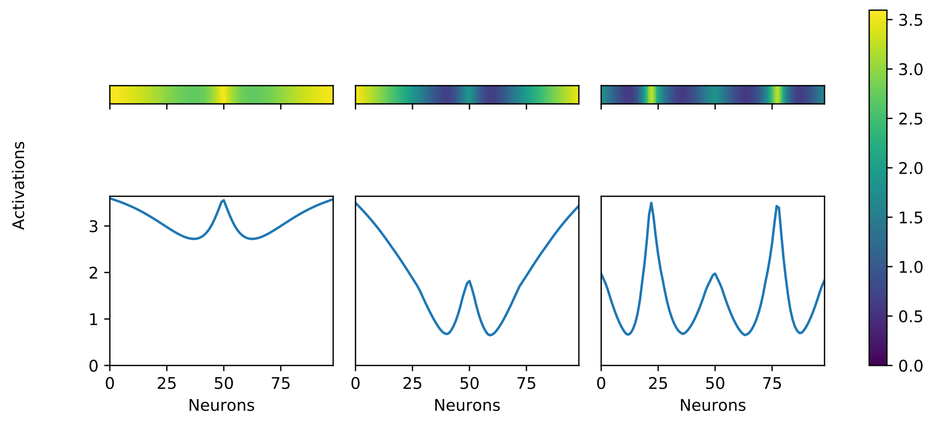


Figure 3.27: Activations of the 40 receiptive fields of the first 2D convolution. The input is convoluted by the 40 kernels, each one produces an array of 99 elements which is filtered by the ReLu activation. The images are activations of the CNN when the input is a datapoint with $w = 0, 1, 2$ respectively.

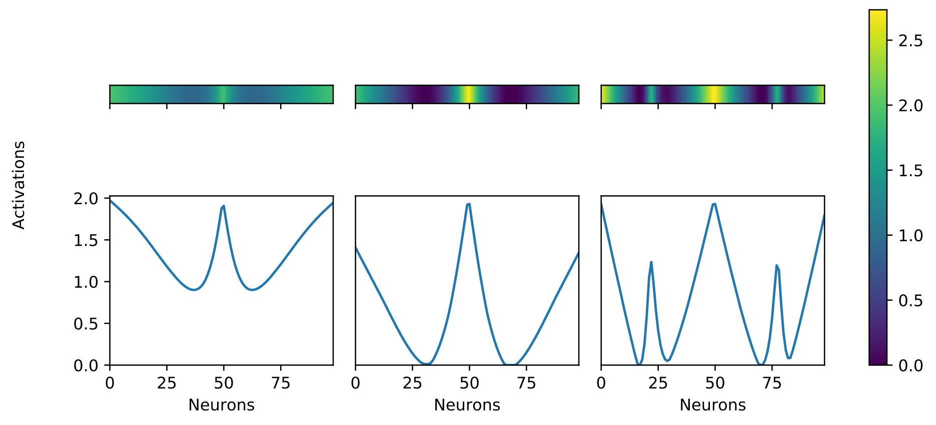
ated. The second convolution brings the results of the 99 rows of the activations in Figure 3.27 together. That happens thanks to a linear combination. This result is passed through ReLu activation. The output is a single array of 99 values with different intensities which resembles a smooth curve. We present the activations of the second layer in Figure 3.28. Here, for every dataset $\mathcal{D}_i$ we plot the curves produced for an input datapoint of phase 0, 1 and 2 going from left to right. For every curve plotted there is also a colored line to represent it on top of it. That will be helpful to understand the interacting data results. The color representation helps to see the curve in a more compact way. The three "curves", on each line of Figure 3.28, are different from each other and this is very helpful for classification since they become the input to a Fully Connected Neural Network with one hidden layer and then the output. Notice also how they are on very different scales since the ReLu activation does not normalize the output. In particular the first and second curves (produced for an input datapoint of phase 0 and 1 respectively) are similar in shape but different in shape. The CNN probably exploits this difference for classification. We infer that it is very easy for the DNN part to recognize such different inputs and that is why the classification of the CNN is so efficient. In order to produce such different curves the ConvNet is taking into consideration the shape of both correlations $C(k)$ and $F(k)$.



(a) $\mathcal{D}_1$



(b) $\mathcal{D}_2$



(c) $\mathcal{D}_3$

Figure 3.28: Ativations of the second convolutional layer for the non-interacting data.

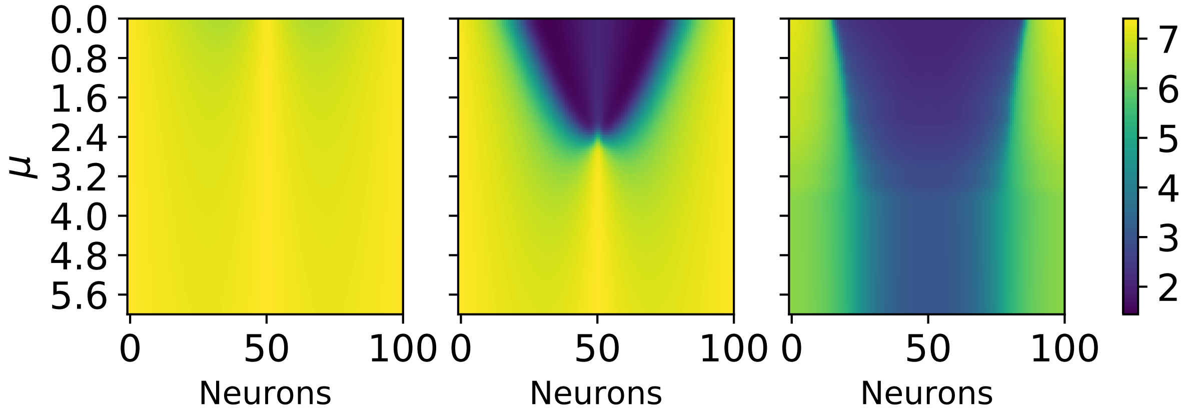
Interacting Data We show now the activation of the second layer, right before being fed to the fully connected part of the algorithm, for the interacting data. This is done in Figure 3.29 where we show the activations for different values of V, with varying μ. Each line of Figure 3.29 presents the results of a different dataset. In each row, from left to right we plot the activations due to the interacting data at potential $V = -2, 0.2, 6$ respectively. For a specific V and μ there are 99 values of the second layer activation. So each panel shows 32 activation lines corresponding to different values of $\mu \in [0, 6.2]$ (vertical axes) for a fixed V. Each row of Figure 3.29 has the same scale, shown in the colorbar to their right.
From this plots we can see the input to the last fully connected part before the classification. Let us start from the low potential values:

- $V = -2$. The activations shown in the plots of the left column are very similar. They resemble the activations in Figure 3.28a of the typical input with winding 0. In fact the CNN has no trouble classifying this points correctly. Notice how every model trained with a different dataset has a different scale for this values.

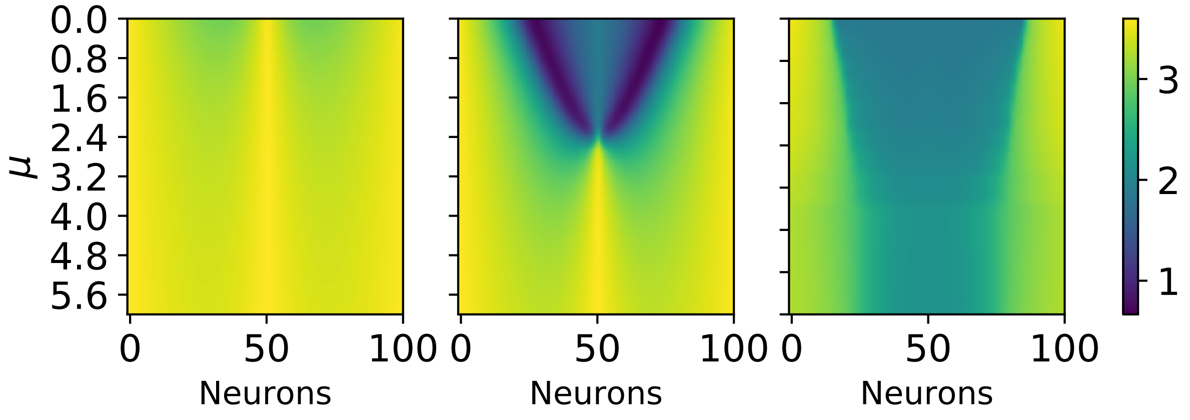
- $V = 0.2$. The activations of the central column show that the input points change phase from superconducting non trivial (phase 1) to superconducting trivial (phase 0). Indeed for μ in the range $[0, 2.4]$ the activations are similar to the ones in Figure 3.28b of the phase 1 non interacting points. After μ hits the phase transition point the shape of the activations changes abruptly to the ones of the figures on their left. That is in fact the same topological sector. Once again we see how the ConvNet perfectly classify these datapoints.

- $V = 6$. The right column of the figure shows the most interesting behaviour. The activations do not seem to resemble the ones in Figure 3.28a of phase 0. They indeed look more alike to the activations of a datapoint with winding 1. Having said that, the CNN classify these points accurately as belonging to the trivial phase 0.

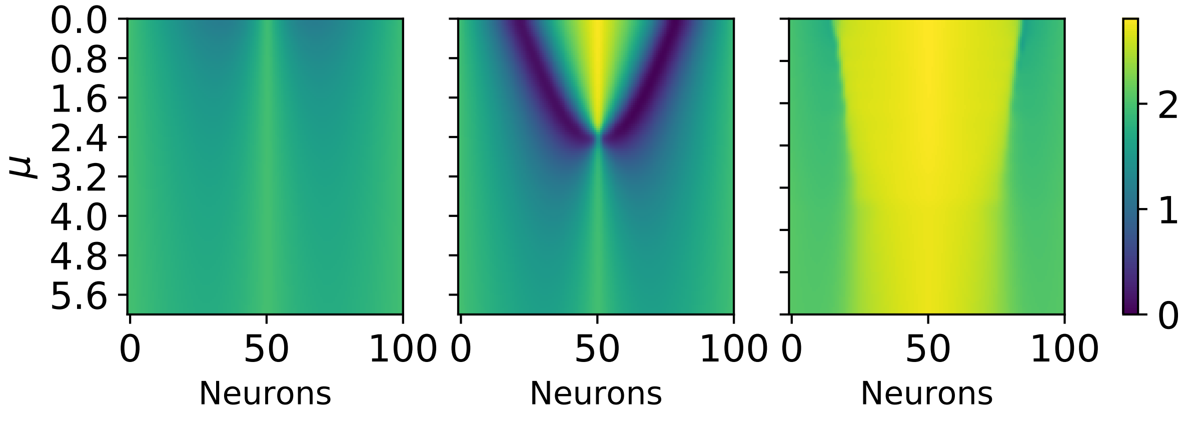
What is remarkable about these results is that the CNN seems to recognize that the datapoints at low and high potential are different. For low potential, the activations are similar to what we expect for a point in the superconducting trivial phase. For high potential, the activations correspond to a different new behaviour that we could expect for the Density Wave sector. Yet, it assigns the same topological phase to both since they are indeed both trivial phases due to the absence of edge modes.
If we compare these results with the activations of the neural network 3.20 we see



(a) $\mathcal{D}_1, V = -2, 0.2, 6$



(b) $\mathcal{D}_2, V = -2, 0.2, 6$



(c) $\mathcal{D}_3, V = -2, 0.2, 6$

Figure 3.29: Activations of the second convolutional layer of the CNN. The three rows are for the different models trained on the datasets $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3$. These show the values of the 99 elements fed as input to the last neural network part of the CNN for different values of $V = 6, 0.2, -2$ (from top to bottom figures) and $\mu \in [0, 6.2]$ (for each figure).

that both do not change at high potential for $\mu \in [0, 6.2]$ but the DNN simply classifies with the wrong phase every data point.

Conclusions In conclusion of this section we can make some reasonings on the results. Firstly, we showed through the PCA how the non-interacting data are easier to classify than their interacting counterpart. Secondly, we saw how even a well trained Fully Connected Neural Network which perfectly recognises non-interacting data has indeed problems in the recognition of interacting ones. We associated this problem to the fact that it does not take into account the shape of the input. Therefore, the CNN seemed the best option to choose and in facts proved to be solid in the recognition of the phases of the interacting Kitaev chain, allowing us to draw the phase diagram from scratch with only the knowledge of the non-interacting model.

Conclusions and Outlooks

In this thesis we applied Machine Learning and Deep Learning procedures and algorithms to study the phases and the phase transitions of the Interacting Kitaev Model in one dimension using correlation functions of the non-interacting Kitaev Model.
We set up the problem taking into consideration the results from the Principal Component Analysis. This decomposition showed us the difference in the complexity of our datasets: the non-interacting one has more defined phases, easier to cluster than the interacting ones. From this we expected to find the correct phase of the interacting model with more difficulty. In fact, the Neural Network we trained on non-interacting data was able to classify with high accuracy and certainty only a part of the interacting phase diagram.
In particular the DNN seemed to have problems in distinguishing the high potential, trivial phase, apart from the superconducting low dimensional one. Opening up the "black box", checking the weights and activations of neurons, indeed showed that this network concentrate on some local features of the correlation functions, such as the presence or absence of peaks at values around π and 2π.
These first results suggested the development of a second type of network: the Convolutional Neural Network. Thanks to their convolution of the input data, seen as a matrix of values, this type of network can take into account the spatial distribution of the training data. This was used by the ConvNet to perfectly classify the right phase of every datapoint of the interacting case. That allowed us to reconstruct the phase diagram of the interacting model with an optimal degree of accuracy with the only knowledge of the non-interacting case.
This result is very interesting in the field of Condensed Matter Physics in which many models do not have analytical solutions that allow to plot the phase diagram exactly. In addition to that we were able to give a sensible estimation of what happens inside of neural networks and which processes led them to make a decision

when assigning the topological phase to a point.
This work can be extended making some more considerations. Firstly, in terms of PCA we only used the first three principal components, while the data showed that much information was contained in remaining components, in particular for the interacting model. To further analyze this data one would need to reduce again the dimensionality of the representation, for example using clustering methods such as t-distributed Stochastic Neighbor Sampling (tSNE) which are keen to work on dimensional reduction of high-dimensional data.
Another extension of this work would be to employ different sets of data. For example including the density-density correlation. This would make the input of the CNN a 3×100 matrix, with more information in it. On the same line of reasoning, one could consider employing the values of the Hamiltonian on the momentum space for different values of the discretized momenta as it is done in [14]. Another option of dataset could be the entanglement spectrum which encodes the topological behaviour of a model.
In addition to that, it could be considered to use different models on our data. For example Random Forests for classification, which have a high interpretability potential, or one dimensional convolution on the correlation functions interpreted as time series.
Lastly, the procedure we implemented in this thesis might be reproduced on many other models of which we do not know the behaviour of when interaction is added to the model.

Bibliography

[1] K. v. Klitzing, G. Dorda, and M. Pepper. New method for high-accuracy determination of the fine-structure constant based on quantized hall resistance. *Physical Review Letters*, 45(6):494–497, August 1980.

[2] Andrei Bernevig and Titus Neupert. Topological superconductors and category theory, 2015.

[3] Andreas W. W. Ludwig. Topological phases: classification of topological insulators and superconductors of non-interacting fermions, and beyond. *Physica Scripta*, T168:014001, Dec 2015.

[4] A. Yu. Kitaev. Unpaired majorana fermions in quantum wires. *Physics-Uspekhi*, 44(10S):131–136, Oct 2001.

[5] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[6] Francois Chollet. *Deep Learning with Python*. Manning Publications Co., USA, 1st edition, 2017.

[7] Giuseppe Carleo, Ignacio Cirac, Kyle Cranmer, Laurent Daudet, Maria Schuld, Naftali Tishby, Leslie Vogt-Maranto, and Lenka Zdeborová. Machine learning and the physical sciences. *Rev. Mod. Phys.*, 91:045002, Dec 2019.

[8] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, and David J. Schwab. A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124, May 2019.

[9] Giuseppe Carleo and Matthias Troyer. Solving the quantum many-body problem with artificial neural networks. *Science*, 355(6325):602–606, Feb 2017.

[10] Giacomo Torlai and Roger G. Melko. Learning thermodynamics with boltzmann machines. *Physical Review B*, 94(16), Oct 2016.

[11] Lei Wang. Discovering phase transitions with unsupervised learning. *Physical Review B*, 94(19), Nov 2016.

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

[13] Evert P. L. van Nieuwenburg, Ye-Hua Liu, and Sebastian D. Huber. Learning phase transitions by confusion. *Nature Physics*, 13(5):435–439, Feb 2017.

[14] Pengfei Zhang, Huitao Shen, and Hui Zhai. Machine learning topological invariants with neural networks. *Physical Review Letters*, 120(6), Feb 2018.

[15] Ning Sun, Jinmin Yi, Pengfei Zhang, Huitao Shen, and Hui Zhai. Deep learning topological invariants of band insulators. *Physical Review B*, 98(8), Aug 2018.

[16] Xie Chen, Zheng-Cheng Gu, and Xiao-Gang Wen. Local unitary transformation, long-range quantum entanglement, wave function renormalization, and topological order. *Physical Review B*, 82(15), Oct 2010.

[17] M. Nakahara. *Geometry, topology and physics.* Institute of Physics Publishing, 2003.

[18] Andreas P. Schnyder, Shinsei Ryu, Akira Furusaki, and Andreas W. W. Ludwig. Classification of topological insulators and superconductors in three spatial dimensions. *Physical Review B*, 78(19), November 2008.

[19] Michael Victor Berry. Quantal phase factors accompanying adiabatic changes. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 392(1802):45–57, 1984.

[20] Di Xiao, Ming-Che Chang, and Qian Niu. Berry phase effects on electronic properties. *Reviews of Modern Physics*, 82(3):1959–2007, Jul 2010.

[21] Yuezhen Niu, Suk Bum Chung, Chen-Hsuan Hsu, Ipsita Mandal, S. Raghu, and Sudip Chakravarty. Majorana zero modes in a quantum ising chain with longer-ranged interactions. *Physical Review B*, 85:035110, Jan 2012.

[22] Martin Leijnse and Karsten Flensberg. Introduction to topological superconductivity and majorana fermions. *Semiconductor Science and Technology*, 27(12):124003, Nov 2012.

[23] N. N. Bogolyubov, V. V. Tolmachev, and D. V. Shirkov. A New method in the theory of superconductivity. *Fortschritte der Physik*, 6:605–682, 1958.

[24] Iman Mahyaeh and Eddy Ardonne. Zero modes of the kitaev chain with phase-gradients and longer range couplings. *Journal of Physics Communications*, 2(4):045010, Apr 2018.

[25] Elliott H. Lieb, Theodore Schultz, and Daniel Mattis. Two soluble models of an antiferromagnetic chain. *Annals Physics*, 16:407–466, 1961.

[26] J. P. Keating and F. Mezzadri. Random matrix theory and entanglement in quantum spin chains. *Communications in Mathematical Physics*, 252(1-3):543–579, Oct 2004.

[27] A. R. Its, F. Mezzadri, and M. Y. Mo. Entanglement entropy in quantum spin chains with finite range interaction. *Communications in Mathematical Physics*, 284(1):117–185, Aug 2008.

[28] Wade DeGottardi, Manisha Thakurathi, Smitha Vishveshwara, and Diptiman Sen. Majorana fermions in superconducting wires: Effects of long-range hopping, broken time-reversal symmetry, and potential landscapes. *Physical Review B*, 88(16), Oct 2013.

[29] Hosho Katsura, Dirk Schuricht, and Masahiro Takahashi. Exact ground states and topological order in interacting kitaev/majorana chains. *Physical Review B*, 92(11), Sep 2015.

[30] Ronny Thomale, Stephan Rachel, and Peter Schmitteckert. Tunneling spectra simulation of interacting majorana wires. *Physical Review B*, 88:161103, Oct 2013.

[31] David Barber. *Bayesian Reasoning and Machine Learning.* Cambridge University Press, USA, 2012.

[32] Y. LeCun, P. Haffner, L. Bottou, and Y. Bengio. Object recognition with gradient-based learning. In D. Forsyth, editor, *Feature Grouping.* Springer, 1999.

[33] Nitish Shirish Keskar, Dheevatsa Mudigere, Jorge Nocedal, Mikhail Smelyanskiy, and Ping Tak Peter Tang. On large-batch training for deep learning: Generalization gap and sharp minima, 2016.

[34] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145–151, January 1999.

[35] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2014.

[36] F. Rosenblatt. Perceptron simulation experiments. *Proceedings of the IRE*, 48(3):301–309, March 1960.

[37] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, October 1986.

[38] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, *The Handbook of Brain Theory and Neural Networks*. MIT Press, 1995.

[39] Vincent Dumoulin and Francesco Visin. A guide to convolution arithmetic for deep learning, 2016. arXiv:1603.07285.

[40] Sebastian J. Wetzel. Unsupervised learning of phase transitions: From principal component analysis to variational autoencoders. *Physical Review E*, 96(2), Aug 2017.

[41] Sam Foreman, Joel Giedt, Yannick Meurice, and Judah Unmuth-Yockey. RG-inspired machine learning for lattice field theory. *EPJ Web of Conferences*, 175:11025, 2018.

[42] Lars Onsager. Crystal statistics. i. a two-dimensional model with an order-disorder transition. *Physical Review*, 65(3-4):117–149, February 1944.

[43] François Chollet et al. Keras, 2015. Software available from keras.io.

[44] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard,

Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[45] Roberto Rubboli. Wavefunctions and correlations of the complex kitaev model on a finite chain, 2019. https://amslaurea.unibo.it/19466/.

[46] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.