UNIVERSITÀ DI PARMA

UNIVERSITÀ DEGLI STUDI DI PARMA

DOTTORATO DI RICERCA IN FISICA
CICLO XXXIII

# Studi di machine learning: applicazioni in realtà aziendale e in fisica delle transizioni di fase


Coordinatore:
**Prof. Stefano Carretta**

Candidata:
**Fabiana Cescatti**

Supervisori:
**Prof. Francesco Di Renzo**
**Dott. Giovacchino Tesi**


**Anni 2017/2020**

# Studies in machine learning: applications in business and physics of phase transitions.

**Fabiana Cescatti**

*«Fame coacta vulpes alta in vinea*
*uvam adpetebat, summis saliens viribus.*
*Quam tangere ut non potuit, discedens ait:*
*"Nondum matura est; nolo acerbam sumere."*
*Qui, facere quae non possunt, verbis elevant,*
*adscribere hoc debebunt exemplum sibi.»*
*(Fedro)*

# Contents

# List of Figures

# List of Tables

# Concerning this PhD thesis

This PhD thesis is the result of a collaboration between the University of Parma and Energee3 srl, an information and communication technology company (ICT) based in Reggio Emilia whose core business are consulting, system integration and digital services.

Such a collaboration may make the academic world meet business enterprise in the common field of methods in ICT research: high quality research projects create results that can lead to commercial gain.

The meeting point is the application of machine learning algorithm. The goal is to develop complementary competences building on a common methodological base.

Machine learning algorithms will be investigated aiming both at an academic project and at implementing a company tool. The different approaches of the interacting partners may lead to an exchange of expertise and, as a consequence, to mutual growth.

The main results obtained during the PhD will be presented in the following chapters. Chapter one will present a general overview of machine learning, focusing on neural networks algorithms that will be used as a tool in both the academic and the business projects. Chapter two will demonstrate how machine learning can be used in the study of phase transitions. In particular, a methodological protocol will be developed to avoid any a priori knowledge of the problem in the choice of training sets. Chapter three will discuss the construction of a linguistic tool based on innovative machine learning approaches to natural language processing. The result is *Soundlike*, a specific solution to a customer business requirement.

CHAPTER 1

# Introduction

Contents



In this chapter, we will present the general concepts of machine learning. Starting from the identification of the situations in which these algorithms can help, we will describe what learning means, also from a mathematical point of view. This can introduce the reader to the terms and concepts of machine learning. We then focus on a specific architecture, the feedforward neural network, an example of which will be presented. This very simple architecture will be our workhorse. Basics and mathematical foundations of these can be find in many books, e.g. [1], [2], [3], [4] or [5]. During my corporate training I took advantage of the materials of [6] or [7] (actually there are many online courses available on this subject). For my academic project I make ude of Matlab neural network toolbox; Matlab Help Center provides itself a lot of useful documentation.

## 1.1. Machine learning: the big picture

### 1.1.1. Beyond taxonomy

Machine learning extracts knowledge from data, building mathematical models on them. Usually it is regarded as a sub-category of artificial intelligence

research field, whose general purpose is to built systems able to solve problems. An old approach to problem solving uses hand-coded conditional rules ("if ... else ..." ) to process data. In such a framework, the developer effort comes down to the implementation of a complete set of dichotomous keys which in principle could deal with all the possible input situations. In many cases this procedure is at least very hard to implement. For example, it's hard to identify the characteristics which may lead to distinction of a dog from a cat or a cat from a tiger in a picture. Even if we could find all the formal rules to say "in the picture there is a cat, a tiger or a dog", once we formalized them in an application, we should have to construct a new program to cope with the identification of a lion. Transposing this example in a more general form, we can say that the hand-coded approach has at least two disadvantages:

i We must know exactly and a priori the solution of the problem we are studying, i.e. for every possible combination of inputs, we know the correct output.

ii We can't tackle a different task without redesigning the entire application.

The machine learning approach is different from the previous: it automates a decision making process by generalizing from known examples. In a typical case this would mean:

i we have some historical data, i.e. a reference collection of correct decisions; in other words, we have pairs of input and corresponding output. For example, in the previous case of animal recognition, we have some animal pictures for which we know whether it is a tigers, a dog or a cat;

ii we can infer the existence of a pattern between input and respective output; we know that in every picture there is a given animal and the recognition process amount to a map between an image and an animal class;

iii while this is conceptually clear, we can not pin down mathematically the map relevant to the case at hand; we will instead look for a general framework which could adjust to fit many applications.

So "learning" means using a set of observations to uncover an underlying structure, finding a way to produce the desired output from a given input without specific rules, but adapting a model to observed data by tunable parameters. After fitting the model on our examples, we can ask the output for a new (never seen before) input.

There are many examples of machine learning applications, very different from each other.

- A typical example is the zip code identification . If we collect many handwritten digits, in most cases, we are able to identify the correct values. Implementing a hand-coded rules program to solve this task is not so easy, because we do not have a unique and clear way to recognize them: for example, we could count the black pixels of the handwritten line vs the white background or we can count the number of crossing lines or the number of loops. Machine learning simplifies things since it requires only the collection of pictures labelled by their own digits.

- A popular example is the Iris species classification: it is easier to collect the dimensions (length and width) of sepal and petal for a large number of setosa, virginica and versicolor Iris flowers, rather than checking and confirming a built-in-advance taxonomy. We will discuss this example later.

- Our email spam filter probably works on machine learning algorithms: observing what a single user regards as spam, it can try to guess the class of an incoming email. The power of this filter is the extreme customization which can be reached.

- Discovery customer behaviours using checkout lists or fidelity cards can help to organize customer segmentations into groups with similar tastes. This is useful to arrange items in a supermarket: items commonly bought together can be placed close to each other; companies are interested in recommendation systems, suggesting the same type of products to the same class of customers.

- The example of tiger, dog or cat recognition can be generalised to object and event detection in photographs or videos. In the medical field, the cancer diagnosis is performed by looking at images, so that the presence of a neoplasm can be detected by a machine learning algorithm trained on tomographic images.

- On social media and blogs, there is a wide quantity of chats, messages, posts and articles that can be used to identify the hate factor and, more in general, to perform sentiment analysis.

I would be ready to gamble that the newcomer could have gained some intuition on machine learning after reading the previous examples. This is right the spirit of it: building on a set of known example, we can build knowledge to classify new data.

### 1.1.2. Different classifications

All the previous examples, and machine learning algorithms in general, can be classified into two different types:

i supervised learning: given training examples of inputs (data) and corresponding outputs (class labels), the machine learns how to produce the correct output for new unlabelled data;

ii unsupervised learning: given only inputs as training, the machine needs to find structure or pattern itself in order to produce an output for new data.

In this thesis, we will focus always on supervised learning, if not differently specified. Supervised learning problems can be further grouped into two different types, depending on the output:

i classification tasks, if we have a discrete output domain and we want to predict a class label;

ii regression tasks, if the output variables are real or continuous values.

In the followings, we will present the general mechanism of a machine learning architecture, namely neural networks in a framework of a supervised classification task.

## 1.2. Learning from data

Since all the learning task depends on the provided data, it is fundamental that the examples are as much representative as possible. In particular, before using a machine learning algorithm, it's important to have enough data which are adequately distributed in the configuration space for the considered problem for the problem at hand. Needless to say, one must avoid manifestly wrong examples and contradictions. A model learnt from data is generally as good as the data from which it has been derived. So, a good data set should be large, as correct as possible, consistent and well balanced. In order to build a model to make predictions, we have to select from data the attributes that will affect the desired output, those having no relevance for predictions being classified as noise. The attributes one chooses are called *features*, the output is the *label*, the *target* or the *class*.

The data we use for the learning process can be classified as follows.

The *training set* collects the examples used for fitting (or training, in fact) the model. This amounts to adjusting tunable parameters. The evaluation of the discrepancy between the model predictions ant the correct outputs on this set measures the learning level of the model on the training set itself. This does not necessary mean that the model will work properly on other data.

The *test set* collects data for which we do know the correct output, but they do not enter the fitting procedure. They are used a posteriori (i.e. after the training process is completed) to test how well the model can make predictions on unknown data.

The *validation set* is a third dataset which is used to tune the model's hyperparameters during the training step (a typical example are exit conditions for the fitting algorithm). We could say they are not relevant for fitting the model parameters, but for adjusting the parameters entering the fitting procedure.

Train, test and validation sets must be samples of the same phenomenon, they must have the same distribution function and they must be independent.

Having different dataset sets allow us to define different errors, i.e. different measures of the discrepancy between model predictions and corrects outputs provided by historical records. Trying to perfectly fit the training set (i.e. driving the training error to zero) will at some point unavoidably make the test errors larger. This is what we call *overfitting*, i.e. focusing too much on the details of the training data and loosing contact with the general features. As we said, a machine learning algorithm has some parameters which have to be empirically optimized: usually the larger the number, the more complex the model. The use of a too simple model is known as *underfitting* and in this case we have bad performances on both training and test set. All in all, we have to reach out an equilibrium between a good fit and a good generalization properties: the validation set can help up with this. This set provides information which goes beyond what is provided by the test set: it is used not only to check the validity of the model, but, based on this, also to tune the hyperparameters of the algorithm. The trade off between underfitting and overfitting is reached when the accuracy on training and validation set are closer to each other.

With the previous definitions, we can outline a typical machine learning process as follow

1. Divide data in training, validation and test (as a rule of thumb 70%, 15%, 15%).

2. Initialize parameters to given values and repeat

   i fitting the model on training set;

   ii checking the forecasts on validation set.

   Until the optimization of parameters makes the model good enough in terms of accurate predictions on the validation set.

3. Check performances on the test set.

As mentioned before, each step can hide some dangers, so we always have to pay attention. In step one, we have to take care on how we construct the 3 datasets. it is important that each one has a sufficiently large number of the different classes we want to predict. This means that for the training set we want to learning every possible class. On the other side, for validation and test we need to correctly quantify the performance of the model on the different classes. The second is the core of the learning procedure. In particular, we have to be careful to find a stable solution of the optimization problem, i.e. not to end up in a relative minimum in the space of parameters configurations. In the third step, we must make sure that we have an accurate measure to quantify the generalization ability of our model. All these dangers are mentioned in the following.

## 1.3. Formalization

From a mathematical point of view, we can see the components of the learning process as follow. We consider a $d$-dimensional vector $\vec{x}$ as input, and its corresponding output $d'$-dimensional $\vec{y}$, which can be discrete or continuous values depending on classification or regression tasks. The components of $\vec{x}$ represent the features, the components of $\vec{y}$ are typically the classes we want to classify.

In order to solve a task, we assume the existence of an unknown function $f$ which maps all the possible inputs into their outputs (this is what we previously called "the pattern between input and respective output" in 1.1). We do not know all the elements in the domain of $f$, but we have only a sub-sample of it: our data set is the collection of $N$ input-output pairs, represented by

$$(\vec{x}_1, \vec{y}_1),\ (\vec{x}_2, \vec{y}_2),\ \cdots\ , (\vec{x}_N, \vec{y}_N) \tag{1.1}$$

where

$$\vec{y}_i = f(\vec{x}_i) \tag{1.2}$$

$f$ is called *target function*. *Hypothesis* is the name of a second function $g$ that we get to approximate $f$ as well as possible, depending on our data set. We what $g$ such that

$$g \sim f \tag{1.3}$$

A function $g$ has a double dependence; it depends on $\vec{x}$ and on a set of parameters $\vec{k}$. The training procedure need to fix the parametric dependence of $g$ (i.e. to select a given $\vec{k}_0$) such that

$$g(\vec{x}_i, \vec{k}_0) = \vec{y}_i \qquad \forall i \tag{1.4}$$

Under the assumption that our data (training, validation and test set) are a good sample for all the predictions we would like to make, this provides a meaning for the statement $g \sim f$. We work under the assumption that if $g(\vec{z}, \vec{k}_0) = \vec{w}$, then $f(\vec{z}) \simeq \vec{w}$, where $\vec{w}$ is now a prediction of the model. Finding the final hypothesis, i.e. $g(\vec{x}, \vec{k}_0)$ function of $\vec{x}$ only, requires to minimize an error measure between each $\vec{y}_i$ and $g(\vec{x}_i)$ on training and validation sets. In general, we can use different algorithms to fit the same family of hypothesis functions, although every situation can suggest the implementation of a specific algorithm to solve the task. The union of the hypothesis set and the learning algorithm represents the *learning model*.

Having learned the relationship between input and output on training and validation sets, we can use points in the test set as novel inputs to evaluate the accuracy of the model. The typical definition of the *accuracy* is

$$\text{accuracy} = \frac{\text{number of correct predictions}}{\text{total number of predictions made}} \tag{1.5}$$

This definition of accuracy works well when we have a good balance in our training/validation/test dataset and we are not in presence of rare events. As for the first observation, consider the case of a binary classification and only a 3% of examples belonging to the second class; our model will learn how to predict only the first class. Having a test set with the same probability distribution of the training set, our accuracy will be around the 0.97 value. When we test the same model on a different test set, for example with 70% of data points belonging to first class and 30% belonging to the second one, the accuracy will crash to 0.7.

On the other hand, if we deal with the detection of some rare events, the accuracy can not be a reliable measure of the performance, since it is an average across all cases and we want to focus on some specific events in a multitude of others. Moreover, the accuracy does not take into account different costs of wrong classifications. To have a complete idea of model performance, we can measure the so-called *confusion matrix*, whose columns represent the instances in a predicted class, while in the rows we count the instances in an actual class (or vice versa). In a binary classification task, the matrix element $\mathcal{C}_{ij}$ represent the number (or frequency) of the patterns belonging to the class $i$, classified as belonging to class $j$. If we aim at detecting the class 1, we can pin down the confusion matrix which has the meaning

$$\mathcal{C} = \begin{pmatrix} TP & FP \\ FN & TN \end{pmatrix} \tag{1.6}$$

where

$TP$ true positive,
the samples of class 1 predicted as belonging to class 1

$FP$ false positive,
the samples of class 1 predicted as belonging to class 2

$FN$ false negative,
the samples of class 2 predicted as belonging to class 1

$TN$ true negative,
the samples of class 2 predicted as belonging to class 2

The accuracy metric is the sum over the diagonal of the confusion matrix, divided by the number of total predictions. In some situations we need to minimize the number of a specific misclassified samples rather then the other. For example in diseases prevention tests it is preferable to have a false positive rather than a false negative. In these type of problems, $C$ is a better metric than accuracy.

## 1.4. Neural networks

Neural networks are a family of machine learning algorithms. We work with the feedforward neural networks, also called *multilayer perceptrons*, and "feedforward" concerns the flow of informations through the network, from an input to its prediction, without backward feedback loops.

We can imagine the network as a directed graph with links connecting different layers. A set of functions connects each layer with the following one, each nodes collecting an output from the incoming links and providing it as an input to the outgoing links.

In general, we can write

$$\text{output}(\vec{x}) = g(\vec{x}) = g^{(h_N o)}(g^{(h_{N-1N})}(\dots(g^{(h_{23})}(g^{(h_{12})}((g^{(ih_1)}(\vec{x}))))))) \quad (1.7)$$

where $g^{(ih_1)}$ is the function mapping the input layer to the first hidden layer, $g^{(h_{12})}$ is the function mapping the first hidden layer to the second,and so on. $g^{(h_N o)}$ is the final function mapping the $N$-th layer to the output layer. $N$ is the number of the hidden layers; in Fig1.1, for example, we have only 2 hidden layers.

Usually, every $g^{(ab)}$ function is a composition of two steps:

i a linear function in applied to the previous layer incoming signal, typically a weighted sum;

ii a non linear function, which is called activation or transfer function, is applied to the result of (i)

afterwards the signal pass to the next layer. Every input signal is a collection of features, i.e. it has a vectorial representation. On each layer we also have vectors. The linear function mapping a 4 component input signal to a 6 component hidden layer (see Fig.1.1) can be encoded in a $4 \times 6$ matrix $\boldsymbol{W}$. Afterwards, the non linear transformation plays its role component-wise, i.e. every nodes apply it to its scalar value. The symbol $g^{(ab)}$ hides a vectorial notation, so we can refer to a single $g^{(ab)}$ component in a subscript notation, e.g. $g_k^{(ab)}$.

We can consider the easiest network as a single hidden layer and follow step by step the transformation of input to its predicted output, in order to understand how the learning process happens. In this situation, we have only 3 steps to take:

$$\text{output}(\vec{x}) = g(\vec{x}) = g^{(ho)}(g^{(ih)}(\vec{x})) \tag{1.8}$$

The linear function in $g^{(ih)}$ acts as

$$\boldsymbol{h} = \boldsymbol{W}^T \vec{x} + \boldsymbol{c} \tag{1.9}$$

$\boldsymbol{W}$ is the weights matrix of the linear transformation and $\boldsymbol{c}$ are the biases. Before the hidden units are used as the inputs of the second layer, the non-linear activation function is applied element-wise, as mentioned in (ii).

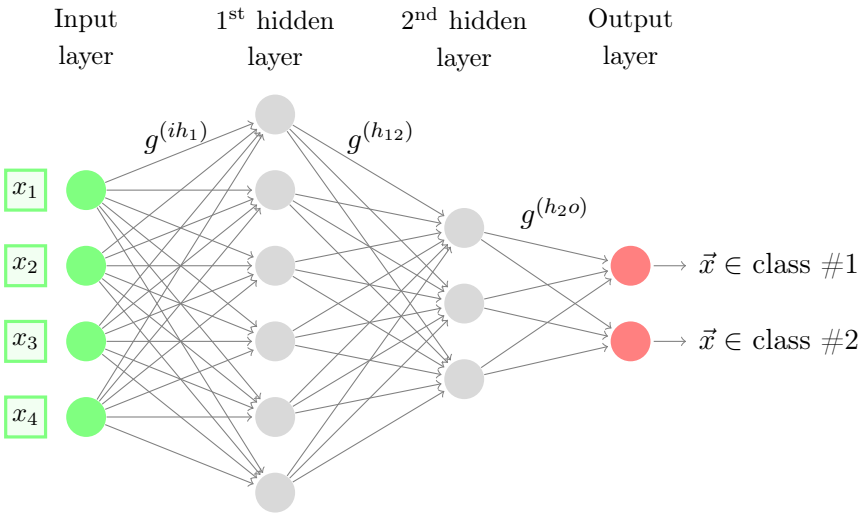



**Figure 1.1:** Schematic representation of a fully connected feedforward neural network with 2 hidden layers. The gray lines an circles represent a black box, i.e. operations which cannot be accessed externaly once the training is finished.

Popular choices of the non linear function are the rectified linear (ReLU) [8] or hyperbolic tangent frequently used

$$g_j^{(ih)}(\vec{x}) = \tanh([\boldsymbol{W}^T\vec{x}]_j + \boldsymbol{c}_j) = \frac{e^{2([\boldsymbol{W}^T\vec{x}]_j+\boldsymbol{c}_j))} - 1}{e^{2([\boldsymbol{W}^T\vec{x}]_j+\boldsymbol{c}_j))} + 1} \tag{1.10}$$

These quantities are passed as inputs to the output layer which performs its linear combination (i)

$$\boldsymbol{z} = \tilde{\boldsymbol{W}}^T\boldsymbol{g}^{(\boldsymbol{ih})} + \boldsymbol{b} \tag{1.11}$$

and applies the transfer function (ii). Softmax functions are often used as final activation functions when the learning task is a $L$ class classification; the rationale is they transform their incoming signal into a probability distribution over $L$ different classes

$$g_l(\vec{x}) = g_l^{(ho)}(g^{(ih)}(\vec{x})) = \frac{e^{\boldsymbol{z}_l}}{\sum_l e^{\boldsymbol{z}_l}} \tag{1.12}$$

This quantity is the model prediction associated to input $\vec{x}$.

Having the correct output for every input training points, we can define a cost (or loss) function which drives the optimization of the tunable model parameters, i.e. the weights and the biases. Due to the non-linear functions introduced as activation functions, we don't have a simple solution coming from linear regression. Typically, we minimize the cost-function making use of an iterative gradient-based optimization procedure, starting from an initialization of random weights and zeros biases. The cross-entropy cost function of the $k$-th output neuron has this form

$$C_k(g,t) = C(g_k,t_k) = -\frac{1}{N}\sum_{\vec{x}}[t_k\ln(g_k) - (1-t_k)\ln(1-g_k)] \tag{1.13}$$

while the total cross-entropy of the neural net is

$$C(g,t) = -\frac{1}{N}\sum_k\sum_{\vec{x}}[t_k\ln(g_k) - (1-t_k)\ln(1-g_k)] \tag{1.14}$$

where $g_k = g_k(\vec{x})$, while $t_k = f(\vec{x})$. Both in Eq. 1.14 and Eq. 1.13 we average over all the $N$ input examples; in Eq. 1.14 we also sum over all the output nodes. The closer the predictions to the correct targets, the closer to zero the cross-entropy, with $C$ vanishing for an exact result. Therefore, the aim during the training is to minimize $C$.

We can do this by updating the weights $\tilde{\boldsymbol{W}}_{ij}$ and the biases $c_j$ with a gradient-descendent optimization algorithm on cost-function and updating the weights $\boldsymbol{W}_{st}$ and the biases $b_t$ by "back-propagating" the cost through

all the non-linear activation functions of the network. Since the gradient descendent is a local procedure, the minimum reached depends on the path followed on the surface in the parameters configurations space; in other words it depends on the order in which the different training examples are passed to the neural network. It is possible to feed the net with the same data but in a different order to readjust the model weights in order to reduce the error further and further. We call an *epoch* a complete cycle through the whole training dataset, whose elements are used to make one forward pass and one backward pass in the parameters optimization procedure.

Since the learning is an iterative process, we need to know how to stop the training. In the following we list some stopping criteria.

- The first "exit condition" is to impose a maximum number of epochs (how large depends on the problems and it is not easy to fix a priori).
- Another strict threshold is a minimum value of gradient descendent algorithm (min_grad): when the performance gradient becomes smaller than the minimum gradient, stop the learning.
- On the other hand, we can ask to stop the learning when a certain degree of accuracy is reached.
- Looking at the learning process itself, it is possible to define another stopping rule. As mentioned before, the whole dataset is split in the training, the validation and the test sets. At the end of each epoch, the model can be evaluated on the validation data points: stop the training when its cross-entropy does not improve for more than $M$ (or max_fail) consecutive epochs.

When we train a neural network all this quantities are taken into account, which ever comes first stops the training.

All the presented concepts are not an exhaustive review of neural networks algorithms, but they are all the recipes that we will use in the following sections.

## 1.5. An example: Iris classification

The above mentioned concepts can be presented in a standard machine learning example implemented in the Matlab neural network toolbox. As previously mentioned, the Iris dataset is a collection of sepal lengths in cm, sepal widths in cm, petal lengths in cm and petal widths in cm of 150 Iris flowers. Every sample is labelled by its class, represented by one of 3 possible Iris species. Our task is a 3-class classification problem, so the target data are

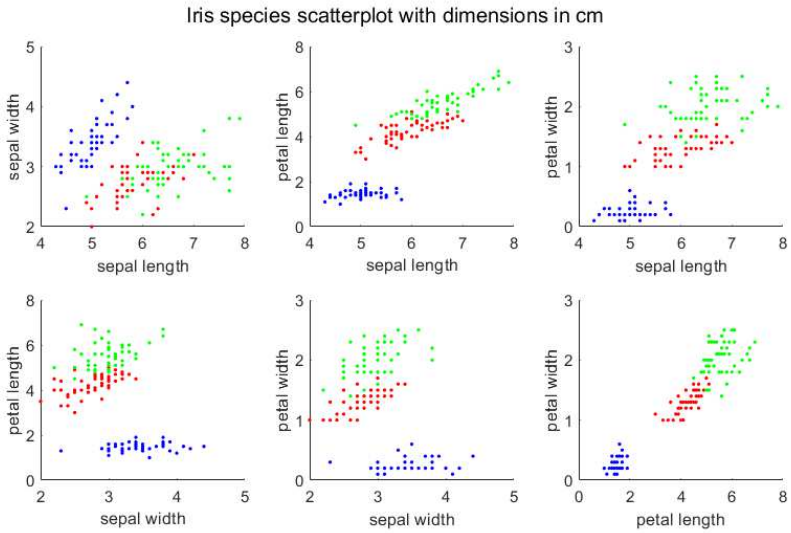



**Figure 1.2:** Iris dataset scatter plot among all pair combination of features. The colours encoded different species.

vectors of all zero values except for a 1 in the class the data belongs to; one such vectors are called one-hot-encoding vectors. They are represented in Tab.1.1.

| | class 1 | class 2 | class 3 |
|---|---|---|---|
| setosa | 1 | 0 | 0 |
| virginica | 0 | 1 | 0 |
| versicolor | 0 | 0 | 1 |

**Table 1.1:** Iris species target data.

Fig.1.2 visualizes the distribution of Iris in 6 panels combining different pairs of dimensions and encods the species in 3 different colours. We can see that a hand-coded rule to classify the flower is not trivial.

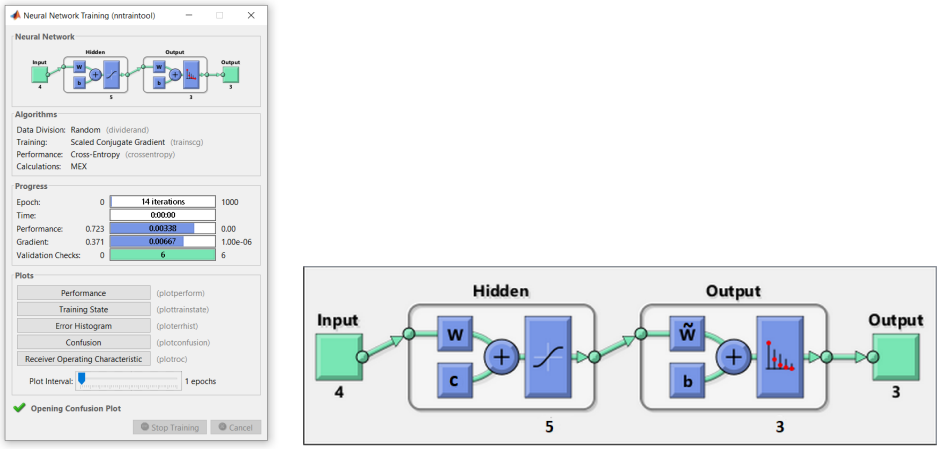
We can implement the following architecture for a simple neural network:

the input layer has 4 nodes, one for each flower feature;

the output layer has 3 nodes, one for each Iris species;

we choose a single hidden layer with 5 nodes.

We can start training keeping the default choices for options and parameters. In particular, the cost function is the cross-entropy, the activation function of the hidden layer is the hyperbolic tangent and of the output layer is the




(a) (b)

**Figure 1.3:** (a) The Matlab training control window; (b) A zoom on the top of the left panel, where we can see a schematic representation of the architecture which highlights the 2 steps structure of every $g^{(ab)}$ function in blue boxes.

sofmax function. Matlab automatically divides the dataset in the 3 datasets, training, test and validation when the train is called.

Matlab provides a training control window, Fig.1.3 (a). The upper part is zoomed in Fig.1.3 (b). In this figure we visualize the schematic architecture of the network (see also Fig.1.1) where blue boxes allude to the $g$ steps: (i) linear function and (ii) non linear function applications. Some details of the training algorithm are monitored in a Progress table, where we can inspect in real time the evolutions of "exit conditions" of the algorithm.

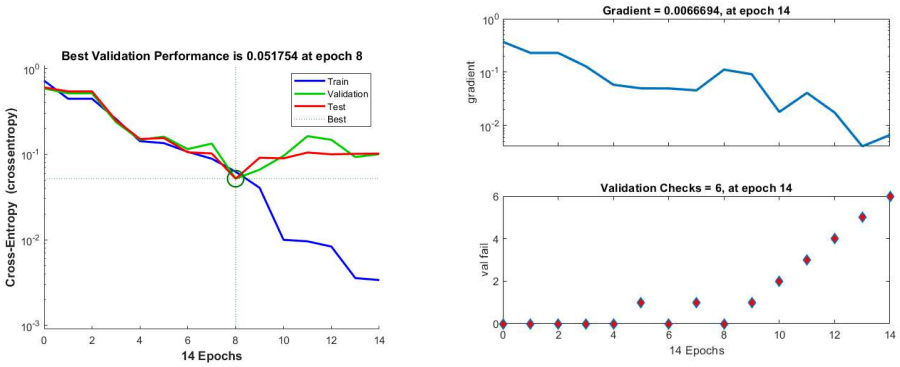
When the train is finished, Matlab also provides plots which we can see in Fig.1.4. We can inspect

> the cross-entropy calculated at each epoch for training, test and validation datasets;
>
> the gradient on training set;
>
> the behaviour of performance on validation in terms of increasing or decreasing of cross-entropy function, i.e. number of validation fails.

We can see from control window that the max_fail condition comes first (in teal in Parameter panel control, Fig. 1.3).

Looking at the performance of our model on the training set, we have

$$\text{accuracy} \simeq 0.913 \tag{1.15}$$




(a) (b)

**Figure 1.4:** Evolution of exit conditions.

while all the confusion matrices calculated on training, test, validation and whole datasets are reported in Fig.1.5. We reach a high level of accuracy on test, higher than training. We can see that the hardest specie to recognize is the second class: the algorithm predicts the wrong specie on training examples 9 times out of 37. The high score for accuracy in Eq. 1.15 hides the not so brilliant result encoded in second column of training confusion matrix. This is due to the in internal random split on data: in the test set, in fact, we have fewer examples of the second class than of the third. If in the test we picked an equal sample from each class, we would get a worse accuracy.


**Figure 1.5:** Confusion matrices of our neural net for Iris species classification.

CHAPTER 2

# Machine learning for phase transitions

Contents



In this chapter we study machine learning as a new possible numerical approach to the investigation of phase transitions. A supervised fully connected feedforward neural network is our choice. In particular, we present a methodological protocol to determine the best training set to use for every linear lattice size without any a priori knowledge of the problem.

## 2.1. General concepts of phase transitions

A phase transition is a qualitative change of the properties of a macroscopic system, induced by infinitesimal variations of one (or more) external parameter(s). An exhaustive presentation can be found in [9]. From a mathematical

point of view, phase transitions are associated to a non analytic behaviour of some derivatives of the thermodynamic potential described in the system. Starting from a general Hamiltonian

$$\mathcal{H} = \sum_{n=1}^{N} K_n \phi_n = \sum_{n=1}^{N} K_n \phi[\theta_n] \tag{2.1}$$

with $K_n$ interactions constants, $\phi$ energy operator and $\theta_n$ the degrees of freedom of $n$-th particle of the system, the partition function is

$$Z_\Omega[K_n] = \sum_{\Theta} e^{-\beta \mathcal{H}_\Omega[\Theta]} \tag{2.2}$$

In Eq. 2.2 $\Theta = \{\theta_n\}_{n=1}^{N}$ is the generic configuration of the system; the sum runs over all possible configurations; $\Omega$ is the region occupied by the system. The partition function encodes all the statistical informations of the system and the probability to be in configuration $\Theta$ is

$$P[\Theta] = P(\mathcal{H} = \mathcal{H}[\Theta]) = \frac{1}{Z} e^{-\beta \mathcal{H}[\Theta]} \tag{2.3}$$

$\beta$ being the inverse temperature. Such a probability allows to determine the energy expectation value as

$$\langle \mathcal{H} \rangle = \sum_{\Theta} \mathcal{H}[\Theta] P[\Theta] = \frac{1}{Z} \sum_{\Theta} \mathcal{H}[\Theta] e^{-\beta \mathcal{H}[\Theta]} = -\frac{\partial \ln Z}{\partial \beta} \tag{2.4}$$

The expectation values of all observables are derivatives of the logarithm of the partition function. Starting from the thermodynamic definition of the Helmholtz free energy $F = E - TS$ ($T$ is the temperature of the system), by standard thermodynamic manipulation, $F$ can be defined as

$$F_\Omega[K_n] = -k_B T \ln(Z_\Omega[K_n]) = -T \ln(Z_\Omega[K_n]) \tag{2.5}$$

(Boltzmann constant is set to 1). Every physical quantity is a convenient derivate of $F_\Omega$ with respect to a given $K_n$. Only in the thermodynamic limit one can inspect a non analytical behaviour of some physical quantities (the partition function is a sum of exponential functions). In this limit, the free energy per single particle (or bulk free energy) can be written as a sum of a bulk part ($f_b$) and a surface one ($f_s$):

$$F_\Omega[K_n] = V(\Omega) f_b[K_n] + S(\Omega) f_s[K_n] + O(L^{d-2}) \tag{2.6}$$

The macro informations on the system can be extracted in the limit

$$\lim_{V(\Omega)\to\infty} \frac{1}{V(\Omega)} F_\Omega[K_n] = f_b[K_n] \tag{2.7}$$

As $f_b$ is a continuous function, its derivatives can present some (removable, jump or essential) discontinuities: the phase transitions are connected to these discontinuities.

In a number of cases, the system can be described by an order parameter, a quantity $m$ that across the transition varies from $m = 0$ to $m \neq 0$. There are cases in which recognizing the quantity acting as the order parameter is a non trivial task. In many cases, it is an average value of an appropriate internal variable $y$. In the linear response formalism, we can write the order parameter as

$$\langle y \rangle \sim -\frac{1}{V(\Omega)} \frac{\partial F_\Omega}{\partial K_s} \tag{2.8}$$

$K_s$ being the field coupled to $y$ in the notation of 2.1. Across a first order phase transitions, the order parameter jumps. For a second order phase transitions, susceptibility associated to the order parameter diverges at the critical point:

$$\chi = \frac{\partial \langle y \rangle}{\partial K_s} \sim -\frac{\partial^2 F_\Omega}{\partial K_s^2} \tag{2.9}$$

In the Ehrenfest classification, a phase transitions is of order $i$ when the $i$-th derivative of $F$ is ill defined. In the case of second order phase transitions, the external parameter is often the temperature. In such a case, the transition occurs at the so-called *critical temperature* $T_C$.

In the proximity of $T_c$ the non analytical behaviour of physical quantities, $g(T)$, can be described by a power law characterized by the so called critical exponents. Introducing the reduced temperature as

$$t = \frac{T - T_C}{T_C} \tag{2.10}$$

the critical exponent $\lambda$ is defined as

$$\lambda = \lim_{t \to 0} \frac{\log(|g(t)|)}{\log |t|} \tag{2.11}$$

We expect

$$g(t) = \mathcal{C} t^\lambda (1 + \bar{\mathcal{C}} t^{\lambda_1} + \dots) \tag{2.12}$$

In the next paragraph we will present the system we will take as a test-bed for our studies. It is a remarkable example of a phase transition.

## 2.2. Ising model

The Hamiltonian of the Ising model is:

$$\mathcal{H} = -J \sum_{\langle ij \rangle} \sigma_i \sigma_j - h \sum_{i=1}^{N} \sigma_i \tag{2.13}$$

The spin variable can assume only 2 values $\sigma_i = \pm 1$; $J > 0$ is a ferromagnetic coupling; $h$ is an external magnetic field; interactions are only between nearest-neighbours $\langle ij \rangle$. As $F = E - TS$, the balance between energetic and entropic terms is driven by the temperature. When $T = 0$, $F$ is minimized when all the spins are aligned and equal to 1 or -1 (depending on $h$). We expect an order phase when the energy term is the leading one. As $T$ increases, the entropic term is switched on. We expect a disordered phase when the entropic term is the dominant. In all these, the (Euclidean) dimension of the lattice plays a key role. On $d = 2$ square lattice, a critical point arises when $h = 0$ and $T$ is

$$T_C = \frac{2J}{\log(1 + \sqrt{2})} \tag{2.14}$$

as shown in 1944 by Onsager's exact solution [10].

The order parameter which distinguishes the two phases of the Ising model is the magnetization per spin ($N$ is total number of spins)

$$m = \frac{M}{N} = \frac{1}{N} \sum_i \langle \sigma_i \rangle \tag{2.15}$$

which vanishes in the disordered phase and is $m \neq 0$ in the ordered one. In a general spin model, the total magnetization can be expressed in term of the free energy as

$$M = -\frac{\partial F}{\partial h}\Big|_{h=0} \tag{2.16}$$

while the magnetic susceptibility is defined as

$$\chi = -\frac{\partial^2 F}{\partial h^2} = \beta \langle (\sum_i \sigma_i - M)^2 \rangle \tag{2.17}$$

Close to the critical point, which is located on $h = 0$ axis, as $t \to 0$, we have the following power-law singularities:

$$m = m(h = 0, T) \sim |t|^{\beta} \tag{2.18}$$

$$\chi = \chi(h = 0, T) \sim |t|^{-\gamma} \tag{2.19}$$

where $\beta$ and $\gamma$ are the relevant critical exponents, whose values depend on the universality class of the problem. In the case of $2d$ Ising model, we have $\beta = 1/8$ and $\gamma = 7/4$.

We now define the connected correlation function as

$$\Gamma(\vec{r}_i, \vec{r}_j) = \Gamma_{ij} = \langle \sigma_i \sigma_j \rangle - m^2 \tag{2.20}$$

This relation holds true for a translational invariant system ($\langle\sigma_i\rangle = \langle\sigma_j\rangle = m$). In such a case, $\Gamma$ depends only on the distances $\|i-j\|$. Away from the critical temperature, $\Gamma$ decays as an exponential

$$\Gamma(\vec{r}) \sim r^{-\tau} e^{-\frac{r}{\xi}} \tag{2.21}$$

where $\tau$ is a number and $\xi$ is the correlation length. Roughly speaking, $\xi$ measures the maximal distance at which 2 spins are not independent degrees of freedom. At the critical temperature, a long range order is established, $\xi \to \infty$ and

$$\Gamma(\vec{r}) \sim \frac{1}{r^{d-2+\eta}} \tag{2.22}$$

$\eta$ being yet another critical exponent equals to $1/4$ in the case of $2d$ Ising model. The last critical exponent we introduce is the one which enters the power law for the correlation length behaviour as $t \to 0$

$$\xi(h=0,T) \sim |t|^{-\nu} \tag{2.23}$$

which enables us to write the susceptibility as a function of correlation length

$$\chi \sim \xi^{\gamma/\nu} \tag{2.24}$$

As mentioned before, focusing on the $2d$ Ising model, we have $\nu = 1$.

## 2.3. Finite size scaling

As a phase transition can occur only in the thermodynamic limit, its investigation through numerical simulations is strongly limited by the finite lattice size $L$. In particular, $L$ acts as a cut off for the divergence of the correlation length at $T_C$. In order to overcome such a limitation, we can introduce a scaling ansatz which describes $F$ as a function of both $T$ and $L$:

$$F = F(L,T) = L^{-d}\mathcal{F}(tL^{1/\nu}) + \text{corrections} \tag{2.25}$$

Through this definition, making use of scale invariance arguments, one can derive the following scaling behaviour for total magnetization and susceptibility

$$M = L^{-\beta/\nu}\mathcal{M}^0(tL^{1/\nu}) \tag{2.26}$$

$$\chi = L^{\gamma/\nu}\xi^0(tL^{1/\nu}) \tag{2.27}$$

with $\mathcal{M}^0$ and $\xi^0$ scaling functions. Moreover, on a finite lattice, the presence of an upper bound for $\xi$ at $T_C$ prevents the divergence of the susceptibility. Instead, it will achieve a maximum value at a pseudocritical temperature, $T_C^{(L)}$, described by replacing $L$ to $\xi$ in Eq. 2.24

$$\chi_{max} \sim L^{\gamma/\nu} \tag{2.28}$$

Measuring the susceptibility for different values of $L$, it's possible to extrapolate the critical exponents and critical temperature from the asymptotic behaviour, since $T_C^{(L)} \sim L^{1/\nu}$.

## 2.4. Swendsen-Wang algorithm

The above mentioned procedure is the canonical approach to the investigation of phase transition through numerical simulation on finite lattices. Such investigations are always performed by means of Monte Carlo simulations. Dynamical Monte Carlo methods relies on a simple application of law of large numbers. We trade the computation of thermal averages for the computation of averages over the evolution of a stochastic process whose asymptotic distribution is the probability measure (Eq. 2.3). Being interested in the asymptotic distribution, we need to let the system thermalize till a equilibrium is reached after which we compute time averages for physical quantities we are interested in [12].

More precisely, the idea of Monte Carlo methods is to generate a stochastic process on state space $S$, always a Markov chain, whose equilibrium distribution we call $\pi$. As long as the system has reached its equilibrium, we can measure time averages, which converge to $\pi$-averages for any initial configuration.

A Markov process is a sequence of random variable $X_0$, $X_1$, $X_2$, ... such that successive transitions, $X_n \to X_{n+1}$, are independent. We can specify the Markov chain by defining a transition probability matrix

$$\mathcal{P} = \{p(x \to y)\}_{x,y \in S} = \{p_{xy}\}_{x,y \in S} \tag{2.29}$$

such that $P(X_{n+1} = y | X_n = x) = p_{xy}$. Since $\mathcal{P}$ is a probability matrix, it satisfies $\forall x, y \; p_{xy} > 0$ and $\forall x \; \sum_y p_{xy} = 1$. If we can move from any state to any other, the Markov chain is called irreducible: $\forall x, y \in S$, $\exists \tilde{n} \geq 0$ for which $p_{xy}^{(\tilde{n})} > 0$ where

$$p_{xy}^{(\tilde{n})} = P(X_{n+\tilde{n}} = y | X_n = x) = p_{xy}^{\tilde{n}} \tag{2.30}$$

is the $\tilde{n}$-steps transition probability. An irreducible Markov chain defines a good Monte Carlo for statistical mechanics if its stationary (equilibrium) distribution

$$\sum_x \pi_x p_{xy} = \pi_y \quad \forall y \tag{2.31}$$

is the Boltzmann distribution

$$\pi_x = \frac{e^{-\beta E_x}}{Z} \tag{2.32}$$

A sufficient condition for the stationarity of $\pi$ is the so-called detailed balance condition for $\pi$

$$\forall x, y \in S \quad \pi_x p_{xy} = \pi_y p_{yx} \tag{2.33}$$

A general method for constructing a transition matrix satisfying detailed balance condition for spin systems is the so-called single-spin flip à la Metropolis [11]. The idea is to start from a general irreducible transition matrix, $\mathcal{P}^{(0)} = \{p^{(0)}_{xy}\}$, and to accept the proposed move $x \to y$ with probability $a_{xy}$ (and to reject it with probability $1 - a_{xy}$). If the move is not accepted, we do not any transition, i.e. $x \to x$. The transition matrix $\mathcal{P}$ can be written as

$$\begin{aligned} p_{xy} &= p^{(0)}_{xy} a_{xy} \quad \text{for } x \neq y \\ p_{xx} &= p^{(0)}_{xx} + \sum_{y \neq x} p^{(0)}_{xy}(1 - a_{xy}) \end{aligned} \tag{2.34}$$

and it satifies the detailed balance condition if

$$\frac{a_{xy}}{a_{yx}} = \frac{\pi_y p^{(0)}_{yx}}{\pi_x p^{(0)}_{xy}} \tag{2.35}$$

If $p^{(0)}_{xy} = p^{(0)}_{yx}$, we can write

$$a_{xy} = F\left(\frac{\pi_y}{\pi_x}\right) \tag{2.36}$$

and, for Eq. 2.32, we have

$$\frac{\pi_y}{\pi_x} = e^{-\beta(E_y - E_x)} \tag{2.37}$$

The Metropolis choice for $F$ is the acceptance probability $F(z) = \min(z, 1)$, i.e. we have the following acceptance or rejection rules

- If $E_y - E_x \leq 0$, then we accept the move $x \to y$ with probability equals to 1.
- If $E_y - E_x > 0$, then we accept the move with probability equals to $e^{-\beta(E_y - E_x)}$, i.e. we pick a random number $r$ between $[0, 1]$ and accept the proposal move if $r \leq e^{-\beta(E_y - E_x)}$.

It is well know that single spin flip algorithms are affected by a severe critical slowing-down: we have autocorrelations effects over large simulation time. This is captured by a large value of the integrated autocorrelation time, being $f = \{f(x)\}_{x \in S}$ a real-valued observable

$$\tau_{int,f} = \frac{1}{2} \sum_{t=-\infty}^{+\infty} \rho_{ff}(t) \tag{2.38}$$

where

$$\rho_{ff} = \frac{C_{ff}(t)}{C_{ff}(0)} \qquad \text{with} \quad C_{ff}(t) = \langle f_s f_{s+t} \rangle - \langle f_t \rangle^2 \tag{2.39}$$

$\tau_{int,f}$ controls the statistical error on the quantities we compute, according to

$$\text{var}\left(\frac{1}{n}\sum_{t=1}^{n} f_t\right) \simeq \frac{1}{n}(2\tau_{int,f})C_{ff}(0) \tag{2.40}$$

To avoid these critical slowing-down problems, we simulate the Ising model with $J = 1$ using the Swendsen-Wang (SW) algorithm [13]. Its effectiveness stems from the non-local nature of the updating procedure; it is a prototype of the so called cluster algorithms. The idea is the following

i Given a spin configuration, introduce a "bond" variable, $b \in \{0, 1\}$, for each pair of nearest neighbours spins with the following probabilities

$$\begin{cases} P(b_{ij} = 1|\sigma_i \neq \sigma_j) = 0 \\ P(b_{ij} = 1|\sigma_i = \sigma_j) = 1 - e^{-2\beta} \end{cases} \tag{2.41}$$

That is the bond variable $b_{ij}$ is set to 1 with a probability equal to $1-e^{-2\beta}$ for two nearest neighbours spins such that $\sigma_i\sigma_j = 1$; when $\sigma_i\sigma_j = -1$ the bond is set to zero.

ii Given a bond configuration, clusters are identified: 2 sites are in the same cluster if a continuous path of bonds connects them.

iii Make a spin flip for all clusters with a probability equal to 1/2.

iv Erase the bond variables and you are left with a new spin configuration.

The configurations of the system are described in terms of spin variable only. The bond variables are only introduced in order to perform an evolution step. It can be shown that the method fulfils the requirement of irreducibility and fulfils the detailed balance relation which, as already mentioned, are known to be a sufficient condition for having 2.3 as asymptotic distribution of the process.

## 2.5. Machine learning approach

As a phase transition occurs when the symmetry of the Hamiltonian is spontaneously broken by changing some control parameters, we can build an order parameter to describe the different phases. The standard approach to the study of phase transitions relies on the investigation of the system with respect to this parameter (most often one studies some susceptibility connected to the latter). As we saw, Monte Carlo methods allow to measure all the thermodynamic quantities of a system by stochastic importance sampling of configurations. Unfortunately, it is not always trivial to identify the symmetry which drives the phase transition and so the order parameter is not always known. Recently, many proposals were put forward to make use

of machine learning algorithms to investigate phase transitions.

Classification is a typical machine learning task. With this respect, one can easily argue that there must be ways of tackling phase transitions detection by means of machine learning. Roughly speaking, the problem amounts to classifying phases. In the case of the Ising model, given a set of configurations sampled from the thermal bath by Monte Carlo methods, a convenient machine learning algorithm should be able to recognise whether a given one belongs to the ordered or to the disorded phase. Supervised techniques are widely used, for example support vector machines in [14], [15], convolutional neural networks in [16], [17]; in [18] it has been discussed the application of a multylayer perceptron neural network.

Starting from a very well-known system, like Ising model on a square lattice, it is possible to investigate how machine learning algorithms work, in order to validate how well they can perform in more exotic situations. In [18], a fully connected feed-forward neural network is trained on a dataset of configurations sampled at different temperatures and labelled as ordered or disordered. The article suggests to use an unsupervised method to assign the correct label to each configuration, namely t-SNE [19]. Another option could be to assign the ordered label to configurations sampled in a given range of low temperatures. The disordered label will be in turn assigned to configurations sampled in a given range of high temperatures. The central question is *how to select the convenient ranges of temperatures.*

One should keep in mind that we always work on finite size lattices and the ranges could well be different for different lattice sizes. This is not at all unexpected: the pseudo critical temperature itself that we want to identify is going to be different for different lattice sizes.

We will approach the study of the Ising model transition making use of the simplest architecture: a fully connected feedforward neural network with only one hidden layer made of 10 nodes. All the work will be implemented in the neural network toolbox of Matlab.

The output layer has 2 neurons: the first recognizes the ordered phase assuming values

$$f = \begin{cases} 1 \text{ for the ordered phase} \\ 0 \text{ for the disorder phase} \end{cases} \tag{2.42}$$

(f stands for the Italian "freddo", i.e. cold). The second recognizes configu-

rations sampled from the disordered phase, i.e.

$$c = \begin{cases} 1 \text{ for the disordered phase} \\ 0 \text{ for the ordered phase} \end{cases} \tag{2.43}$$

(c stands for the Italian "caldo", i.e. hot). First of all we need to train the network. This is turns asks for selecting a convenient training set. Once the network is trained, we feed it with configurations sampled in an extended range of temperatures, in particular across $T_C$ as well. This leaves us with two functions $f(T)$ and $c(T)$. Following the criterium presented in [18], we define a pseudo critical temperature $T_C^*$ as the one at which the network is mostly confused, i.e. the two output neurons have equal values, i.e.

$$f(T_C^*) = 0.5 = c(T_C^*) \tag{2.44}$$

On a finite lattice we do not expect 2 step functions; $f(T)$ and $c(T)$ are smooth functions resembling sigmoids.

## 2.6. Our training protocol

In the perspective of using machine learning algorithms to detect phase transitions, it is necessary to avoid as much as possible a priori knowledge. In particular, our goal is to determine the range of temperatures which optimizes the training of the neural networks.

### 2.6.1. Constructing the training set

Our approach to study phase transitions by machine learning is inspired by [18]. We collected configurations sampled in a range $(T_{min} : \delta T : T_{max})$, i.e. temperatures are equispaced. As mentioned before, the idea is to train a neural network with a collection of configurations labelled as ordered or disordered depending on the temperature at which they are sampled. In order to balance the training dataset between the ordered and the disordered configurations, we subsample a low number of temperatures (among all the simulated ones). In particular, our training protocol can be summarized as follow:

i select a "highest low temperature" $T_l^*$ and a "lowest high temperature" $T_h^*$ (in the following called edge temperatures) and two intervals of equal width $I_{T_l} = [T_l^* - \Delta T, T_l^*]$ and $I_{T_h} = [T_h^*, T_h^* + \Delta T]$;

ii $\forall T \in I_{T_l}$ and $\forall T \in I_{T_h}$, sample a number of spin configurations and label them as ordered and disordered respectively;

iii train the neural network with the configurations and their related outputs ([$f$,$c$]) $[1, 0]$, $[0, 1]$;

**Table 2.1:** Details of simulations and trainings: $T$ is the range of temperatures at which we extract 15000 configurations to feed the network; Cnfgs is the number of configurations used at every temperature as training data set; Loops is the number of times we go through the training procedure, i.e. we repeat it Loops times. Each time we restart from the output of the previous step. Technical parameters of the training can be different, e.g. 5+5 means that we take the default values for the first 5 iterations (min_grad=$10^{-6}$, max_fail=6); we switch to more stringent requirements in the last 5 iterations (min_grad=$10^{-8}$, max_fail=50).

| L | $T$ | $T_l^*$ | $T_h^*$ | Cnfgs | Loops |
|---|---|---|---|---|---|
| 100 | 1.80:0.01:3.19 | 2.09:0.02:2.19 | 2.6:0.02:2.7 | 8000 | 5+5 |
| 80 | 1.80:0.01:3.19 | 2.09:0.02:2.19 | 2.6:0.02:2.7 | 8000 | 5+5 |
| 64 | 1.80:0.01:3.19 | 2.09:0.01:2.19 | 2.6:0.01:2.7 | 5000 | 8+3 |
| 40 | 1.60:0.01:2.99 | 1.85:0.02:2.19 | 2.6:0.02:2.8 | 5000 | 5+5 |
| 20 | 1.40:0.01:3.19 | 1.51:0.02:2.19 | 2.6:0.02:3.0 | 5000 | 5+5 |

.

iv feed the trained net with configurations sampled over the extended range of temperatures (so across $T_C$ as well) and look at the emerging signal.

We choose $\Delta T = 9\delta T$, i.e. each interval is made of 10 equispaced temperatures.

### 2.6.2. Neural network variances

The details of simulations and trainings are presented in Table 2.1. For each choice of edge temperatures, we train $N = 20$ different neural networks. We can thus consider two different types of fluctuations of neural network signal. A first variance is computed with respect to the different spin configurations, given a single neural network. A second variance is that of the signal (we mean the signal averaged over the different spin configurations) with respect to the 20 different networks. In the following we present both.

Denoting the $k$-th configuration of spin with $\sigma_k$, we collected $K = 15000$ different configurations for each temperature, $\{\sigma_k(T)\}_{k=1}^K$. Being the $f$ and $c$ neural network outputs such that $f + c = 1$, it is enough to look at one of the two. We can define the signal of the $i-$th network at temperature $T$ as the following average

$$c_i(T) = \frac{1}{K}\sum_{k=1}^{K} c_i[\sigma_k(T)] \tag{2.45}$$

This signal, as an average, has a variance, due to the probability distribution of configurations

$$\sigma_i^2 = \sigma^2[c_i(T)] = \frac{1}{K-1}\sum_{k=1}^{K} [c_i[\sigma_k(T)] - c_i(T)]^2 \tag{2.46}$$

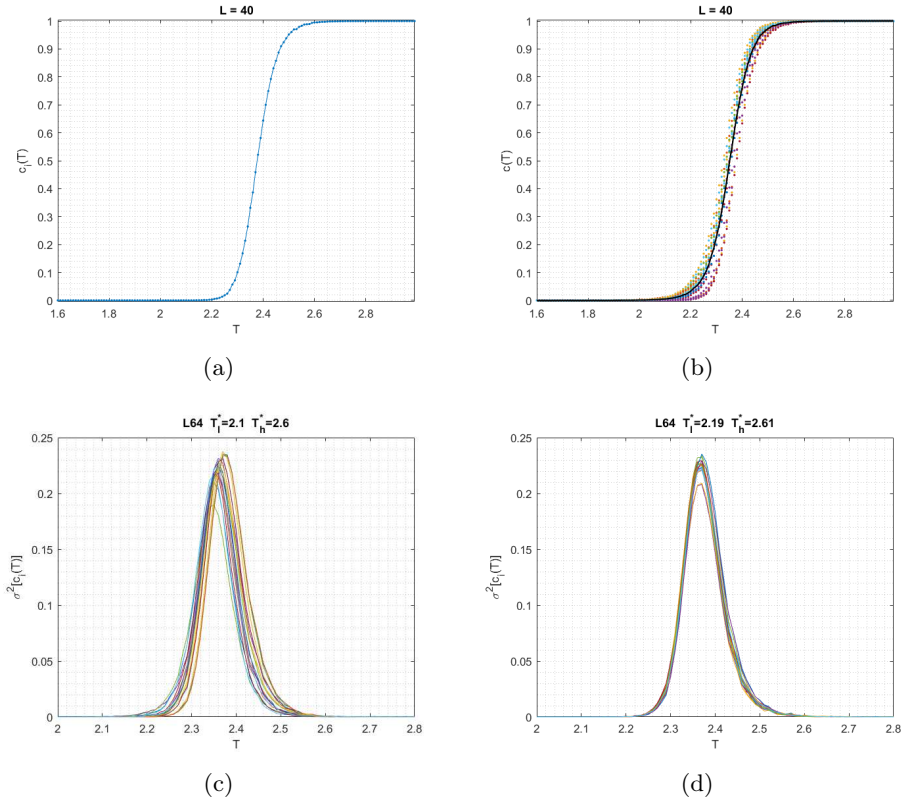



**Figure 2.1:** Panel (a) shows $c_i(T)$. (b) displays $\{c_i(T)\}_{i=1}^{20}$, represented by dots, and $\bar{c}(T)$ represented by a black line. (c) and (d) display $\{\sigma_i^2\}_{i=1}^{20}$ for two different choices of edge temperatures, i.e. different training sets.

Panel (a) of Fig. 2.1 is the typical signal $c_i(T)$ ($i$ denoting a single network). (b) displays 20 signals $\{c_i(T)\}_{i=1}^{20}$, represented by dots. A continuous line represents the average $\bar{c}(T)$ - see below. (c) and (d) are $\{\sigma_i^2\}_{i=1}^{20}$ for two different choices of edge temperatures, i.e. different training sets.

In order to determine a single signal for each edge temperatures at a fixed lattice size, we also average on the $N = 20$ different neural networks

$$\bar{c}(T) = \frac{1}{N}\sum_{i=1}^{N} c_i(T) \tag{2.47}$$

We remind the reader that $\bar{c}(T)$ is the continuous line in Fig. 2.1 (b). A second variance, plotted in Fig. 2.2, is now defined as

$$\sigma^2(T) = \frac{1}{N-1}\sum_{i=1}^{N}(c_i(T) - \bar{c}(T))^2 \tag{2.48}$$

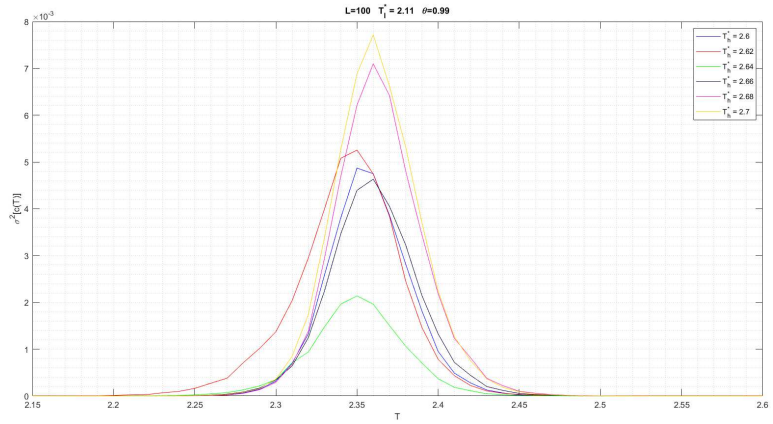



**Figure 2.2:** Variances of the net signals as a function of $T$. $I_{T_l}$ is fixed, while $I_{T_h}$ varies. $L = 100$.

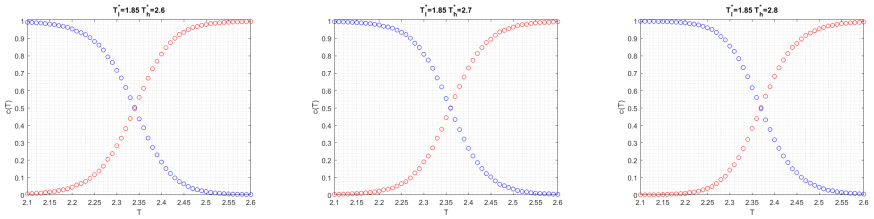
By comparing Fig. 2.1 and Fig. 2.2, one can spot the different order of magnitude, as expected. The variance distribution looks like a Gaussian.

The relevance of the variance defined in Eq. 2.48 will be clear in the following.

### 2.6.3. Selecting best training dataset I: self-consistency score

Let's us come back to the different choices of the edges temperature for the training. In this choice, we want to avoid using any a priori knowledge of the problem. *Our protocol itself will make the selection.* The best temperatures will be selected in force of a *self-consistency argument*. We could say that our model is indeed supervised, but to a certain extent.

So why is the choice of the training set that relevant? Fig. 2.3 shows how different choices of the training set result in different $T_C^*$ values.




(a) (b) (c)

**Figure 2.3:** $T_C^*$ depends on the choice of edge temperatures. In (a) (b) and (c) $\bar{c}(T)$ and $\bar{f}(T)$ are plotted.

We can assign a *self-consistency score* $\mathcal{SC}$ at each edge temperatures in many ways. We want to rank the possible choices in terms of their ability to classify: ideally we would like the training to be as certain as possible in assigning the labels.

i If the disordered label has been assigned to configurations taken at any $T \in I_{T_h}$, we expect that a fortiori we will get $c_i \sim 1$ for all temperatures higher the those in $I_{T_h}$, i.e. $T > T_h^* + \Delta T$. On the other side, if the ordered label has been assigned to each configuration taken at any $T \in I_{T_l}$, we expect that a fortiori we will get $c_i \sim 0$ for all temperatures $T < T_l^* - \Delta T$. We define

$$a = \frac{\delta T}{T_{max} - T_h^* - \Delta T} \sum_{T=T_h^*+\Delta T}^{T_{max}} c_i(T)$$

$$b = \frac{\delta T}{T_l^* - \Delta T - T_{min}} \sum_{T=T_{min}}^{T<T_l^*-\Delta T} [1 - c_i(T)]$$

where $T_{min}$ and $T_{max}$ define the overall temperatures interval of the simulations. Notice that

$$\frac{\delta T}{T_{max} - T_h^* - \Delta T} = \frac{1}{N_{high}} \tag{2.49}$$

where $N_{high}$ is the number of temperature $T > T_h^* + \Delta T$. Then we choose a threshold $\theta \simeq 1$ and discard a network for which

$$\frac{a+b}{2} < \theta$$

(those are rare wild pitches).

ii We then compute the score

$$\mathcal{SC} = \left[ \mathcal{A} \left[ 1 - \bar{c}(T_l^*) + \bar{c}(T_h^*) \right] + \frac{\mathcal{B}}{10} \left( \sum_{T \in I_{T_l}} (1 - \bar{c}(T)) + \sum_{T \in I_{T_h}} \bar{c}(T) \right) \right] \tag{2.50}$$

where

$\bar{c}$ is the average signal of all the networks which satisfy the (i) condition

$\mathcal{A}$ and $\mathcal{B}$ are two tunable normalized weights

In practice, we compute the average signal of the networks at the edge temperatures and the average on training intervals $I_{T_l}$ and $I_{T_h}$; finally we construct a linear combination with tunable coefficients (weights). After some tries, we set $\mathcal{A} = \mathcal{B} = 0.5$.

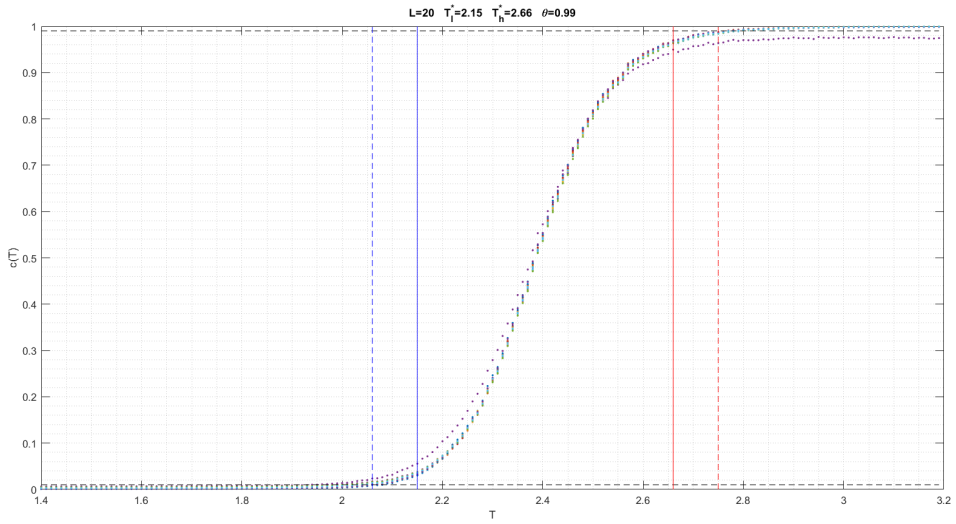



**Figure 2.4:** The quantities used to calculate the self-consistence score are drawn. The dashed black line is the threshold $\theta$; purple dots are an example of wild pitch (i). The intervals $I_{T_l}$ and $I_{T_h}$ are easy to spot.

Fig. 2.4 displays the quantity we've just discussed at work.

The best way to inspect the score distribution is probably a heat map with $T_l^*$, $T_h^*$ as axes, see Fig. 2.5. There is indeed some structures in these results. This is particularly true in the case of small systems.

The best trainings are concentrated in the bottom left area, this does not come as a surprise. We are somewhat cheating: for very low $T_l^*$ and very high $T_h^*$ we will for sure end up with very ordered and very disordered configurations. The point is we want to recognize effective samples of the phases over an interval which is "reasonably" close to the (pseudo) critical temperature. Moreover, fixing the low edge $T_l^*$, the best scores are concentrated in the bottom of the panel while, fixing the high edge $T_h^*$, the best scores are concentrated on the right. Bottom right corner panel of Fig. 2.5 is one way of looking at this. We fix a threshold for the self-consistence score: on the yellow-blue border we find the closest temperatures to the critical region for which the accuracy is at least equal to the threshold. Despite this interesting result, it turns out that this procedure does not suggest a unique choice of the edge temperatures.

### 2.6.4. Selecting best training dataset II: variance score

We then decided to look at a different score, i.e. the variance presented in Eq. 2.51. For a given choice of $(I_{T_l}, I_{T_h})$, $N^*$ is the total number of networks

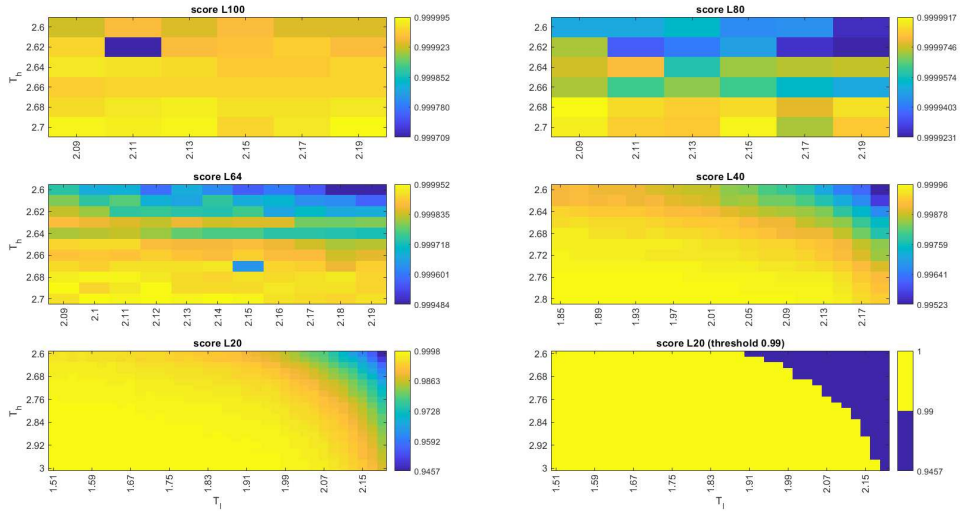



**Figure 2.5:** Self-consistence scores for different values of edge temperatures and different values of lattice linear size. Bottom right, a threshold on scores is enforced.

satisfying condition (i) (i.e. the rare wild pitches are discarded)

$$\sigma^2(T) = \frac{1}{N^* - 1} \sum_{i=1}^{N^*} (c_i(T) - \bar{c}(T))^2 \tag{2.51}$$

As already mentioned, this is a difference variance with respect to $\sigma_i^2(T)$ ( Eq. 2.46). The latter has been used in [20] in a different strategy for the quest of the critical temperature. We remind the reader that in our approach $T_C^*$ is defined as $\bar{c}(T_C^*) = 0.5$, (as suggested in [18]). We notice that all the 3 temperatures (our definition of $T_C^*$, the temperature of the peak of $\sigma_i^2$ and the temperature of the peak of $\sigma^2$) are very close to each other. Our definition of $T_C^*$ can be thought [18] of as the temperature at which the network is *mostly confused in its classification task*. In a similar spirit we will select the training set, i.e. $(I_{T_l}, I_{T_h})$, as the one for which the network is *the least confused it its training task*. As a result, the $N$ neural networks encode the very similar informations.

In Fig. 2.6 we plot the heat map of the maximum values of the variance. Our suggestion is to select as training edges, e.g. as training set, the temperatures at which the maximum of Eq. 2.51 is minimum: the smaller the dispersion, the more robust the learning. With respect to Fig. 2.6, we select the $T_l^*$ and $T_h^*$ temperatures corresponding to the deeper blue cell.

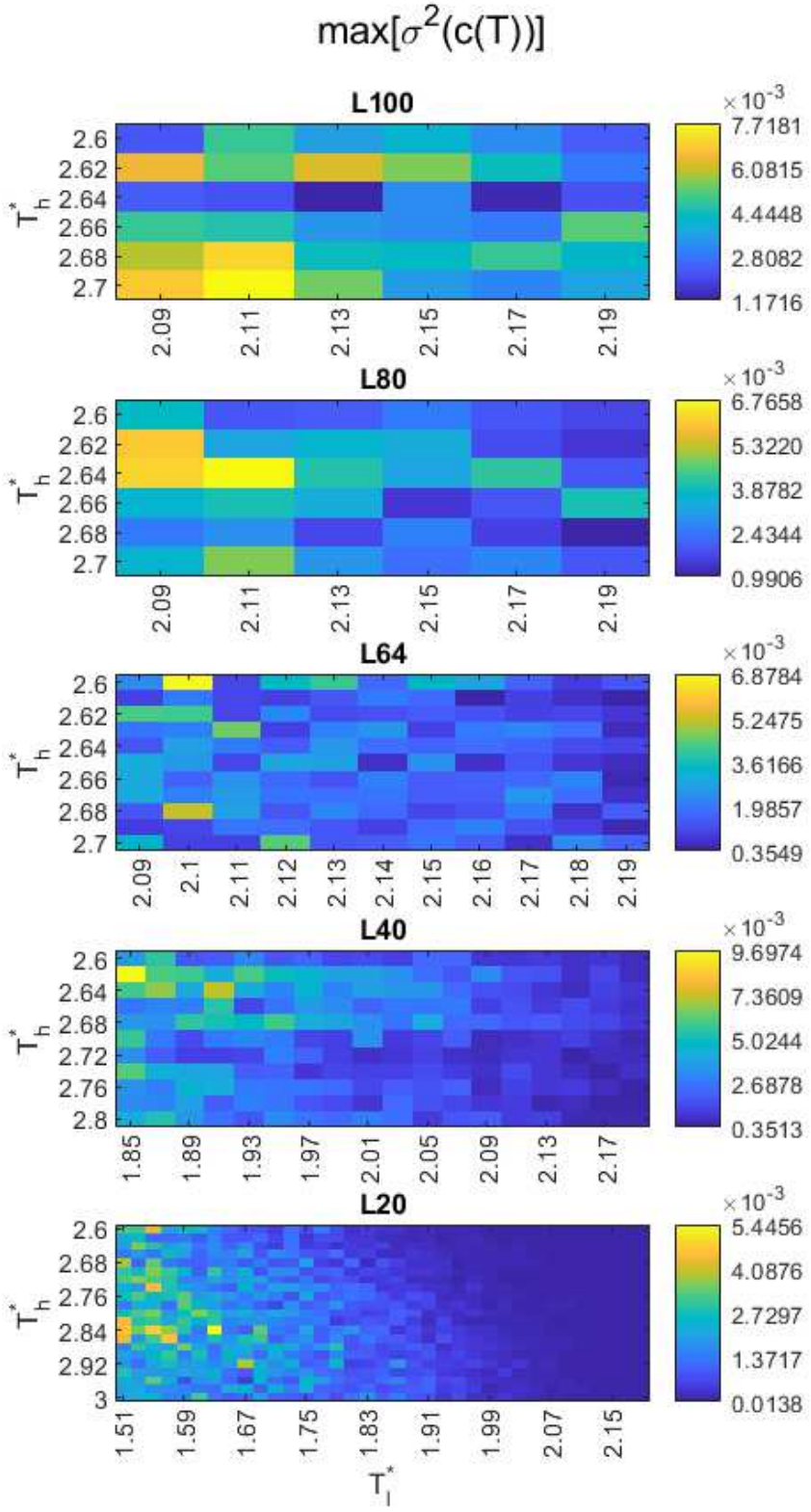



**Figure 2.6:** Heat map of net signals variances, for every lattice linear sizes and for every pairs of edge temperatures.

| $L$ | $\Delta T_l$ | $\Delta T_h$ | $T_C^*$ |
|---|---|---|---|
| 100 | 2.04:0.01:2.13 | 2.64:0.01:2.73 | 2.35(5) |
| 80 | 2.10:0.01:2.19 | 2.68:0.01:2.77 | 2.36(3) |
| 64 | 2.10:0.01:2.19 | 2.61:0.01:2.70 | 2.36(7) |
| 40 | 2.08:0.01:2.17 | 2.78:0.01:2.87 | 2.39(2) |
| 20 | 2.04:0.01:2.13 | 2.78:0.01:2.87 | 2.40(6) |

**Table 2.2:** Training ranges $(I_{T_l}, I_{T_h})$ selected by our learning protocol and the corresponding pseudo-critical temperature.

### 2.6.5. Pseudo-critical temperatures

Once training set is decided, we can compute $\bar{c}(T)$ and look for $T_C^*$: $\bar{c}(T_C^*) = 0.5$. This has been obtained by interpolation (spline function).

The numerical results are presented in Tab. 2.2. The minimum variance at each lattice size $L$ is plotted in Fig. 2.7. $T_C^*$ are plotted as vertical lines. Notice that they are always very close to the maximum of $\sigma^2$. In Fig 2.8 we plot the variances of energy and of the absolute value of magnetization, as obtained by Monte Carlo simulations. Once again, the vertical line marks $T_C^*$. For every lattice size $L$, we find that our estimate always fall right on top of the magnetic susceptibility. This, of course, does not come as a surprise, being the magnetization the order parameter. Still this information has never been provided explicitly. This suggests that the way we proceed can indeed be assumed as a protocol, i.e. we expect that it can be useful in cases in which we miss a detailed understanding of the mechanism driving the transition (and in particular we do not know the order parameter).

If we look at Tab. 2.2, we can inspect no clearcut trend in the selection of $(I_{T_l}, I_{T_h})$ for different sizes. We only see a rough tendency toward smaller $T_h^* - T_l^*$ as lattice size increases. In spite of this, as $L$ increases, the temperatures selected as the pseudocritical ones move closer to the known $T_C$ of Ising phase transition. In other words pseudocritical temperatures appear to display the behaviour that would be eventually captured by a finite-size scaling curve. In a sense, we are once again cheating: since we sit on top of the peak of the magnetic susceptibility, we have no doubt that finite size scaling will show up.

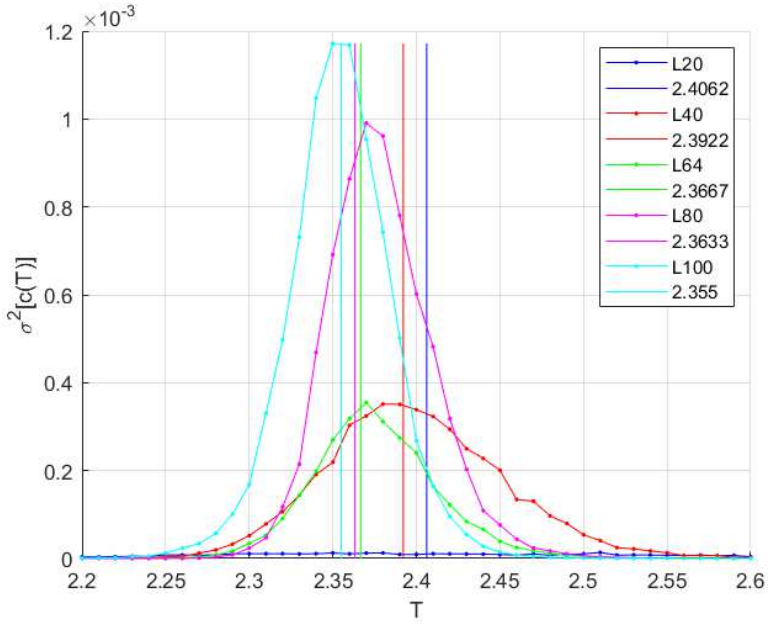



**Figure 2.7:** Variances of the net signals as a function of $T$. For each lattice size, we plot the variance for the selected $(I_{T_l}, I_{T_h})$

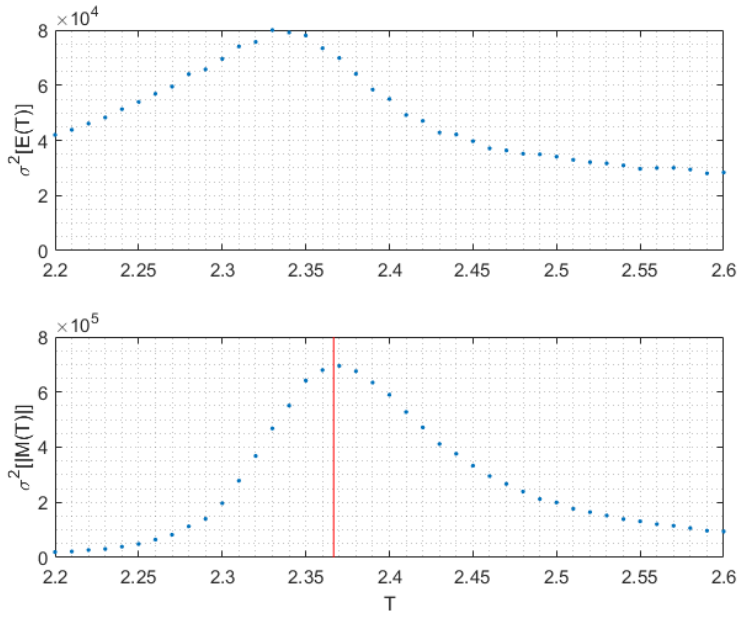



**Figure 2.8:** The variance of the energy and of the magnetization absolute value are compared for $L = 64$ as a function of temperatures. Our $T_C^*$ is drawn with red vertical line.

## 2.7. Conclusion and future perspectives

This work, presented in [21], focuses a methodological aspect of investigating phase transitions with neural networks. In particular, we set up the training set by selecting two temperature intervals in which phases are clearly separated and we propose a protocol for the identification of the best choice of them for every value of the lattice size. All in all, differently from previous literature works, we explicitly test different training sets and choose the best in terms of robust learning. We measure the robustness of learning as minimal variance of the signal of different neural networks trained in the same range.

This study is admittedly only a first step since we have not yet investigated large lattice sizes. Still, we do not expect surprises, given that our $T_C^*$ sits right on top of the magnetization susceptibility.

# CHAPTER 3

# Soundlike

## Contents



In this chapter, we will present the implementation of a documental search engine, based on the natural (Italian) language. It is the main project on which I worked at Energee3. In fact, it represents the study of a business application: the problem is that of a customer asking questions at a bank branch; many times the bank clerk diverts the question to the internal helpdesk. Our goal is to lighten the helpdesk workload and to make the customer waiting time not too long. Since we are dealing with a bank problem,

it is important to be able to understand the technical language of bank sector. This tool has been developed in close collaboration with the ItaliaNLP Lab at the Istituto di Linguistica Computazionale "Antonio Zampolli" (ILC-CNR) in Pisa. In particular, we cooperate with Prof. Felice Dell'Orletta. The laboratory main research line is the study of models, methods and technologies for Natural Language Processing, combining the linguistically-aware data modeling with innovative machine learning approaches [22]. Their expertise has enabled me to understand the concepts of the computational linguistics and the state of the art of this field. Moreover, we could use their sophisticated advanced technology solutions to build our tool.

## 3.1. The problem, the goal and the general concepts

*Soundlike* is a document question answering system written in python language whose aim is to lighten the workload of the bank help desk office. As mentioned in the preface, it is a solution to a specific bank need. The typical situation is: a client asks the branch operator for a specific request; if the operator does not know the answer, he can run a search in the bank internal documentation or he can call the help desk office, which usually know the answer. The second solution is faster and so it is the preferred one in order to reduce the clients' waiting time. The drawback, as already mentioned, is that the the help desk office can not handle the caseload of many callings from all the bank branches. Soundlike can be a solution, if we are able to construct a fast and easy tool for the branch operators to navigate their internal "how to". In order to let the computer replace the help desk, the new application shall be as similar as possible to the current human interaction between branch and help desk. For this reason, Soundlike will be interrogated in Italian natural language, in the same way the operator makes a request on the phone. That's why we have to make the computer a tool able to process human language.

Roughly speaking, as a first request, the tool we are going to set up must be able to understand the words that are typically used in a customer's question. There is of course much more than this, i.e. words are part of documents, and the main goal is right to find the documents that are most relevant to the questions that are asked.

*Computational linguistics* is the field which deals with the computational modelling of natural language and sometimes it makes use of machine learning algorithm for its applications: this is exactly our situation.

### 3.1.1. The word

The *distributional semantics* develops and studies solutions for quantifying the semantic similarities between linguistic items. In particular, the *distributional hypothesis* [23] is the cornerstone of all our models: the word meaning is defined by statistical analysis of its contexts, i.e. the other words which surround it. In other words, we understand the meaning of a word by inspecting how we make use of it in our language: items used in the same contexts purport similar significances. Two items have similar meanings if they occur in the same contexts. In simple terms, an apple is more similar to a banana, rather than to a bicycle, because in the text "apple" and "banana" occurs with the same words like "to eat", "to peel", "fruit ", "snack ", and so on.

In distributional semantics the vectors are the main tools for mathematical representation of the lexical content [24]. All in all, words are mapped to vectors, every vectorial component of a target word encodes the occurrence of the word itself in a specific context. With this in mind, the similarity is a simple geometrical distance between vectors. If we use verbal context to characterize the vector definition of a word, we have to collect a large samples of language data.

A *corpus* is a language resource, that is a large set of texts. We will perform the statistical analysis on those texts. It should be noted that the selection of the language resource plays a key role because it is the starting point of our semantic model on which Soundlike will be designed. In particular, we have to choose carefully the corpus from which we get the information to construct the vectors representation of every word. If we are looking for the representation of the word "run", it makes a difference to use a sport magazine as corpus rather than a business management book.

For our solution, we will investigate two possible choices of corpus. The first one is the Italian corpus from the .it domain (itWaC) [25]. In Sec.3.5 we will present its peculiarities, but for now it is important to know that it collects a general purpose texts by web crawling. The second one collects some texts about banking or financial topics. In this situation, we aim to construct the word vectors for contexts of technical language.

In order to construct a semantic model, one goes through a number of steps [26].

i For every target word, e.g. for every word in our corpus, we count its contexts. We can collect this quantities in a matrix, called *co-occurrence matrix* $\mathcal{C}$. If we have a total of $M$ words, the dimension of $\mathcal{C}$ will be $M \times M$: every word has its own row and columns. The element $\mathcal{C}_{ij}$ represents the number of times the item $i$ occurs in the context $j$. For

example, if in our corpus we count three times "banana" with "eat", we will have, on the row corresponding to "banana" word, the number three in correspondence of the columns of "eat". Of course, the co-occurrence matrix is symmetric.

ii The count of co-occurrence pairs can be replaced by a frequency. Moreover, to highlight the salience of some contexts compared to others, we can move to *statistical weights*, $\mathcal{C} \to \tilde{\mathcal{C}}$. It has been demonstrated [26] that encoding word meaning by context statistical weights, works better then simple counts in different types of semantic tasks. An example of a map from counts to weights could be the point-wise mutual information. It is the logarithm of the ratio of $\mathcal{C}_{ij}$ and the total number of times we count the $i$-th word in the corpus multiplied for the total number of times we count the $j$-th word.

iii Despite the cited transformations, we are typically in front of a sparse matrix. In order to have meaningful vectors representation, a *dimensionality reduction procedure* is applied to $\tilde{\mathcal{C}}$.

iv One ends up with the matrix rows (or columns) as the vector representations of the target words.

In other words, the co-occurrences extracted from corpus are counted, weighted and then dimensional reduced to get some dense word vectors. From a vector of $M$ components, we move to smaller vector whose number of components $N$ can be adjustable. This type of models are called *count models*.

In recent years, a new family of distributional models has been introduced. The *prediction model* use neural networks to create word vectors with fixed dimensionality, without going through the calculation of co-occurrences. It has been proved [27] that, in a very large number of linguistic tasks, prediction models work better than count models; moreover they are more robust. In this field, the representations of target words are called *word embeddings*. The reference algorithm to construct the word embeddings from corpus is word2vec [28], [29], whose mechanism is presented in Sec.3.2.1. We can collect the map from string words to word embedding vectors in a dictionary, called the *word embeddings dictionary*.

As already mentioned, we have to deal with documents and so we have to extend the "word-level" vectorial representation.

### 3.1.2. The document

All the previous concepts are useful to measure the similarity between single words. Our aim is to catch the meaning of sentences, i.e. to measure simi-

larity between groups of words. In fact our task is to quantify a similarity between the branch operator question and the documents in the bank documentation; we know that the latter contain the answer. In other words, the bank documents contain all the possible answers to customer questions.

*The aim is to find among all possible documents the one which maximizes a given semantic similarity with question.*

So, we have to combine the word embeddings of a document (or a branch operator question) in such a way that we get a vector representing the document (or the question) itself. We have to move from word embeddings to a document-level vector representation. The jump from single items to sentences brings along with it an important change. We know that a sentence has it own structure. Although we might analyse the sentence in terms of logical analysis, we rely on *part of speech (POS) tagging*. To be sure to tackle the meaningful documents' words, we collect from each documents only its substantives, adjectives and verbs. Each POS has its own vector representation in terms of word embedding. There is no unique way to combine these word embeddings to get a document. We will follow the procedure suggested in [30]. The *documents embeddings dictionary* contains the matching between the written documents and the document vector representations, called document embeddings. The same procedure can be used to construct the question embeddings, since from a lexical point of view there is no difference: they are both a collection of words.

As for word embeddings, a similarity measure between two documents can be encoded in a geometrical distance between two document embeddings.

In few words:

i word embedding encodes the meaning of a word in a mathematical way (a vector, actually)

ii for each document we combine some selected word embeddings to have a vector encoding the meaning of the whole document; in particular

- we collect from each document adjective, substantives and verbs
- we replace the previous collected words with their word embeddings
- we combine in different ways the previous word embeddings

iii we define a similarity measure

In the following section, we present the computational details of Soundlike building blocks. In Sec. 3.2.1 we describe how to collect word embeddings from corpus by the neural network model word2vec. Having the word

vectors, we define the geometrical distance to measure similarity between two words in Sec. 3.2.3. At the end, in Sec. 3.2.4, we present how to get document embeddings and how the similarity measure can be performed between questions and documents.

## 3.2. The algorithm

### 3.2.1. word2vec

As mentioned before, word2vec is a neural network constructed to identify word embeddings from a corpus. Its schematic representation is in Fig. 3.1. We do not train the net on a specific task in order to make predictions (such predictions are usually read in the output layer). We do train the net on a specific task, but counter-intuitively, we catch the word embedding representations by looking at its hidden layer. In particular, we have the input and the output layers with a number of nodes equal to the total number of (unique) words in the corpus. With this respect we can think of the input (and output) layer as a vector, but notice that the only vectors which we will feed as input (and collect as output) are one-hot-encoding vectors (see the definition in chapter 1), and these are the words in our corpus. To be definite: given an ordering of words, if we feed the third word as input, we will have all the neurons of the input layer set to zero but the third (which is set to one). The hidden layer has a variable number of nodes, the choice of which determines the dimensionality of the word embeddings, e.g. the number of components. From the figure, we see that we have a corpus dictionary made of 4 unique words. Being the hidden layer made of 6 neurons, we can collect the weights in $4 \times 6$ matrix. It will turn out that word embeddings are right the rows of these matrix, i.e. for each of the 4 words we have a 6 components vector.

We remind the reader that two words are in the same context if they are located close to each other in the sentences of the corpus; how close is a tunable parameter called (context) window size ($w$). Given a word $a$, we let the window slide through all the sentences of our corpus. Every single time that we find a word $b$ falls within the window, we established a input-output correspondence between $a$ and $b$. These are the input-output pairs that we will feed to the neural network in the training process. Notice that this is a very large amount of training informations. Notice also that, if we take the point of view of classification neural networks, these informations are even contradictory.

While all the input-output pairs in the training set map a one-hot-encoding vector to a one-hot-encoding vector, this mapping is not the final answer we are interested in. Actually, we want the neural network to map a

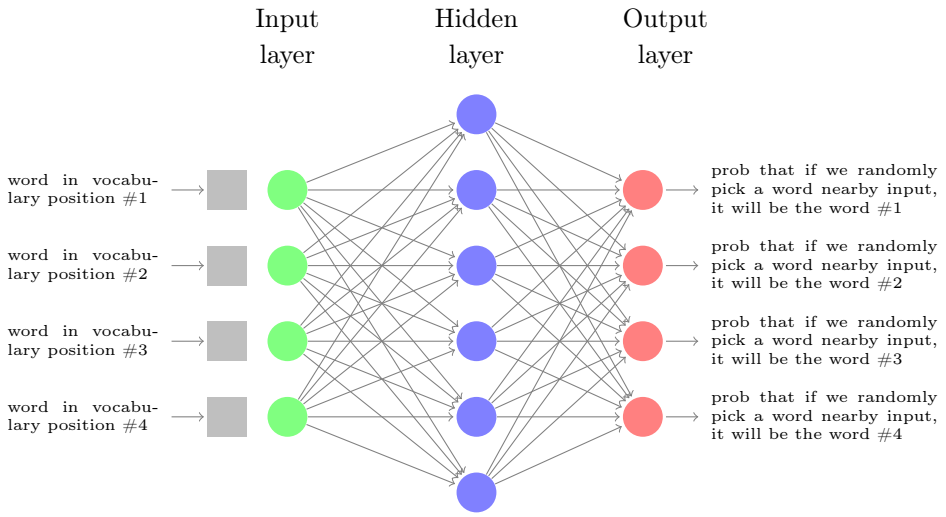



**Figure 3.1:** Schematic representation of word2vec architecture algorithm.

one-hot-encoding vector (i.e. a given word) to a fully-populated output encoding the probability that other words are in its context. All this depends on the context window size.

For example, from the following sentence, with a window size $w = 2$, we collect the following pairs of training data.

"Great Britain is an island."

| | | |
|---|---|---|
| (Britain, Great) | (is, Britain) | (an, is) |
| (Britain, is) | (is, an) | (an, island) |

We now provide an example of the probabilities we expect in the output later once the network has been trained. Given the target word "Great", we expect that the probability of finding "Britain" in its context is larger than the probability of finding "blue"

$$p(\text{“Britain”}|\text{“Great”}) > p(\text{“blue”}|\text{“Great”})$$

where the notation |"Great" means fixing the target word (a conditional probability, actually).

We now come back to a previously made statement: the row of weight matrix are the word embeddings defined by the network. A simple motivation for this is the following: a given word is encoded in a one-hot-encoding vector and thus the output to which the network maps it is entirely fix by one

single row. In our application we look for 64-dimensional word embeddings, so that the hidden layer will have 64 neurons. The magic of the training process is that after the network has been fed with a multitude of (one-hot-encoding, one-hot-encoding), it ends up encoding a (one-hot-encoding, probability) correspondence.

The larger the number of words in the corpus, the larger parameters we have to tune. For this reason, the author of word2vec introduced some tricks to reduce training time without loss of representation accuracy for word embeddings.

The first one is the *sampling* of the most frequent words. It is easy to understand that some items, like articles, appear as contexts for the majority of the words: they don't tell us much about the meaning of the words and, on the opposite side, it's hard to learn a good embedding for them. word2vec allows to set a threshold for occurrence of words; those that appear with higher frequency in the training data will be often discarded in sampling. More precisely, each word $\omega_i$ is discarded with probability equal to

$$P(\omega_i) = 1 - \sqrt{\frac{t}{f(\omega_i)}} \tag{3.1}$$

where $f(\omega_i)$ is the frequency of the word in the corpus and $t$ is a tunable threshold. It has been empirically demonstrated that subsampling accelerates the training time and improves the accuracy of word embedding, especially for uncommon words.

The second trick is called *negative sampling*. In a given training step, the output is a one-hot-encoding and thus has a single entry set to one, all the other entries being zero. In the jargon of the method all this zero as thought as negative labels. In the negative sampling approach out of all the weights relevant to reconstruct the negative labels, we decide to update only a subset, on a random basis. Different experiments show that for small datasets updating a number between 5 and 20 negative labels is enough to get a reasonable accuracy.

Downloading the word2vec code from `https://github.com/dav/word2vec`, it's possible to train the word embeddings as follows:

```
word2vec -train our_words.txt -output vec64.txt -size 64
  -window 5 -sample 1e-4 -negative 5 -binary 0 -cbow 0
```

where

- our_words.txt is the corpus file: it must have the punctuation marks of original texts or breaking lines at the end of sentences;

- size is the dimension of vectorial space that we are going to construct, 64 in the example below;
- window is the window contexts size (it is assumed that window context never slides out of the sentence);
- cbow is a binary parameter: 0 means that we use the skip-gram configuration;
- sample is the tunable threshold which is named $t$ in Eq. 3.1;
- negative is the number of negative label which are updated at each training step;
- vec64.txt is the resulting training file: for each word of the corpus we will find a file row with the word characters string followed by its numerical word embedding components; we will end up with the structure for the word embeddings dictionary.

Given a corpus, the number of known words is fixed. Any word outside the corpus will not be recognized. As a matter of fact, in order to maximize the amount of information it could be useful to minimise the number of entries, working with lemmas rather than words. In this situation, we need a specific tool which transforms every word in its canonical (or dictionary) form; for example *break*, *broken* and *breaking* are all represented by the lemma *break*.

### 3.2.2. Feeding word2vec with lemmas

If we use lemmas as elements, we have to pass the concatenated files to a suitable tool in order to make the lemmatization of words. The tool we use is Text-To-Kowledge (T2K), developed by the ItaliaNLP Lab researchers [31]. T2K returns a CoNLL-U file in which each entry has the following structure:

index | word | lemma | POS | specific POS | features

where

index is the numerical position of the word in the sentence;

word is a given word;

lemma is the canonical or dictionary form of the word (it is not always different from the word itself);

the POS (Part Of Speech) is the grammatical category of the word; notice that T2K takes into accounts much more grammatical structures than the ones we will later consider in the construction of documents (nouns, adjectives and verbs), e.g. a preposition;

specific POS is a more specific POS which best characterizes the grammatical category (typically these are specific categories of Italian language, e.g. a preposition can be a "preposizione semplice" or a "preposizione articolata");

features are the morph-syntactic features, complementing the POS information.

Having recognized a lemma for each word, we have to come back to the original document structure in order to be able to identify the context windows and to construct the word embeddings. The context depends only on the order of the words in a sentence. So, we are allowed to put each lemma at the same positions of its respective word. Now we can concatenate with wrapping the lemmatized corpus and feed it to word2vec.

### 3.2.3. Similarity

Having a word meanings representation by vectors, we can define the similarity between two words as a geometrical distance in our vectors space. In computational linguistic, the most used similarity measure is based on dot product between vectors. Having two non-zero word embeddings, $\vec{v}$ and $\vec{z}$, we define the similarity of the words as the cosine of the angle between them

$$\mathcal{S}(\vec{v}, \vec{z}) = \cos(\theta) = \frac{\vec{v} \cdot \vec{z}}{\|\vec{v}\| \, \|\vec{z}\|} = \frac{\sum_i v_i z_i}{\sqrt{\sum_k v_k^2}\sqrt{\sum_h z_h^2}} \tag{3.2}$$

$\mathcal{S}$ is known as *cosine similarity* and it assumes maximum value of 1 when two embeddings are equals, minimum value -1 when they have “opposite direction”.

Now we have all the building blocks to enable a computer to process a text. In particular, we have the basic concepts to try to understand how an algorithm can solve some linguistic tasks like word analogies, semantic relatedness, named entity recognition, concept categorization, synonym detection and so on.

Back to our problem, we have to find a representation of sentences rather than words.

### 3.2.4. From words to documents

The idea is to have a vectorial representation of the question $\vec{x}$ and all the $N$ documents $D = \{\vec{d_i}\}_{i=1}^{N}$: Soundlike will select as the answer the document

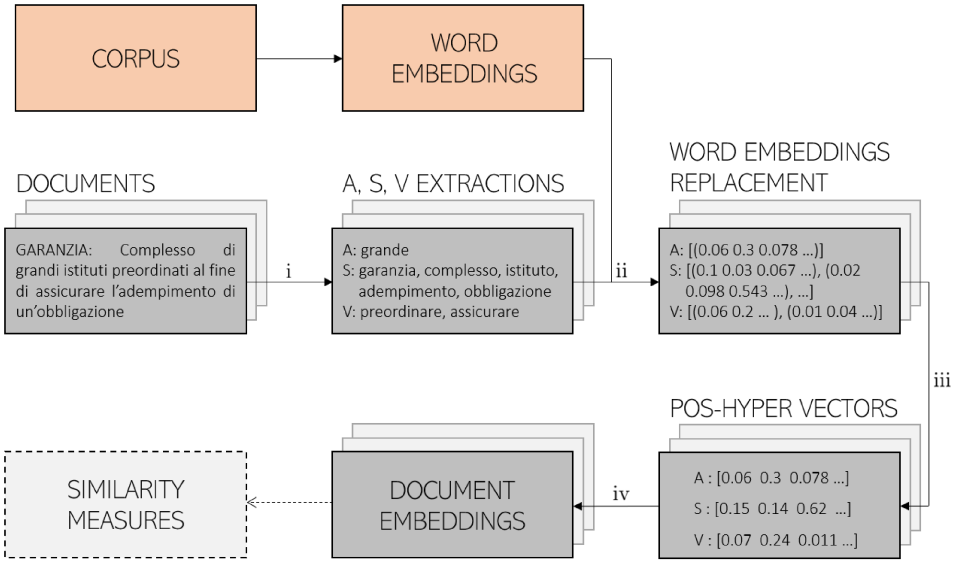



**Figure 3.2:** Schematic representation of document embeddings construction, starting from corpus and textual documents. The orange boxes represent the word-level procedure. In grey boxes, we report the steps to be performed to deal with documents. The POS-hyper vectors are obtained by element-wise sum of word embeddings (in the same POS category). The indexes on arrows are the same of the list in the next page.

(i.e. the vector) which maximizes the similarity

$$r(\vec{x}) = \arg\max_{\vec{d}_i \in D} \mathcal{S}(\vec{x}, \vec{d}) \tag{3.3}$$

As previously mentioned in Sec. 3.1.2, we have to construct the document embeddings from word embeddings. A schematic and complete representation of the procedure is in Fig. 3.2. First of all, we collect from each document only its substantives, adjectives and verbs. To do this we once again use T2K. Given the POS, we replace them with their word embeddings. To construct the document embeddings we now combine the word embeddings of specific POS category. We will call *POS hyper-vector* the result of the combination of the word embeddings for a specific POS category. This in not enough. In order to quantify a single similarity (and not a number of similarities, each for each POS category), we have to make a further choice: either we take into account the cosine similarity of a single POS hyper-vector, having 3 possible choices (adjective hyper vector, substantive hyper vector or verb hyper vector); or we construct a *super-vector* for each document, which combines (in a given way) the POS hyper-vectors. So, we have two different levels of hierarchy: the first is that of POS hyper vectors; the second level is that of super-vector. Notice that one particular rule for obtaining a super-vector is to retain a single POS hyper vector.

All in all, we construct the document embedding through the following steps:

i we collect from documents and questions all the adjectives, substantives and verbs; as we said, the tool T2K, which is able to find lemmas, is also able to perform part-of- speech tagging (POS tagging);

ii we replace each element with its word embedding;

iii we combine the elements of each category of words and, as a result, we have only 3 hyper-vectors for each document: the adjective hyper-vector, the substantive hyper-vector and verb hyper-vector;

iv finally, we need to map each document to a document embedding; to do this, either we retain a single POS hyper-vector or we combine the hyper vectors into a super-vector.

Let us explain point (iii), i.e. how we construct the POS hyper vectors. We can make different choices and consider:

1. means
2. sums
3. max pooling
4. min pooling
5. products

All of them act on the single components of the vectors. For example, if we have a document with 3 adjectives with the following 64-dimensional word embeddings:

$$\vec{a}^{(1)} = (a_1^{(1)}, a_2^{(1)}, ..., a_{64}^{(1)})$$
$$\vec{a}^{(2)} = (a_1^{(2)}, ..., a_{64}^{(2)})$$
$$\vec{a}^{(3)} = (a_1^{(3)}, ..., a_{64}^{(3)})$$

If we make our choice for sums, our adjective hyper-vector will have the following components:

$$\vec{A} = (a_1^{(1)} + a_1^{(2)} + a_1^{(3)}, ..., a_{64}^{(1)} + a_{64}^{(2)} + a_{64}^{(3)})$$

Much the same holds for substantive $\vec{S}$ and verbs $\vec{V}$.

Once we are left with 3 POS hyper-vectors, in order to compute similarities between documents we have to make our choice for step (iv), i.e.

- either we only look at the similarity of a single POS category (we will later refer to these 3 choices as A, B, C in Sec. 3.4);
- or we average 3 similarities that are calculated between hyper-vectors of the same POS; notice that this is a possibility that we have not taken into account (we will later refer to this choice as D in Sec. 3.4);
- or we compute a similarity after constructing an explicit document super vector; this is obtained by concatenating the adjective hyper-vector, the substantive hyper-vector and the verb hyper-vector components (we will later refer to this choice as E in Sec. 3.4).

We have thus defined an overall strategy to tackle our task: at a bank branch a clerk is asked a question and we want to provide her/him with Soundlike, a tool which will select which document she/he has to read to find the answer. In the following we will first of all fill a gap which we left behind and discuss a few details on the corpus construction. We will then proceed to better characterize the workflow of Soundlike in terms of data organisation.

## 3.3. Data organization

### 3.3.1. Collection of the technical corpus

As mentioned before, we use two different corpora: a general purpose one and a technical one.

To construct the technical language word embeddings, we use as corpus both internal bank documents and external documents. The latter are collected by scraping different bank and financial websites, as specified in Tab. 3.1. The data source provided by our customer is a collection of html pages written by help desk office employees for internal use; these are internal bank documents which are supposed to contain possible answers to the questions branch operators are asked.

First of all, we have to extract the relevant text parts from the html documents, navigating the html-tree structure to skip tags and other html elements. In order to facilitate the subsequent analysis, it is necessary to be able to identify the different phrases. Of course one relies on punctuation, but the latter is partially incomplete due to the informal nature of the documents. The structure of html pages, in particular the hierarchy of parent-child tags, can help. For this reason, we assume the following rule: if a html tag changes, the sentence stops. We know that this is not always the case, but we have to provide some general guidelines. If we don't do this, for example when a table is found, we will consider as a unique continuous text

**Table 3.1:** List of websites used as external data source for training specific word embeddings.

| | |
|---|---|
| ABI Banca | Arbitro Bancario Finanziario |
| Arbitro Contorverse Finanziarie | assegni.net |
| Banca Aletti | Banca Akros |
| Banca Cariparma | Banca Centrale Europea |
| Banca d'Italia | Banca Intesa San Paolo |
| Banca Unicredit | Banco BMP |
| Bankpedia | bonifico.org |
| cambiale.net | consap |
| consumatori.it | Credem Banca |
| Misurare e gestire il rischio bancario (ebook) | entrateuscite.org (voci glossario) |
| Ministero dell'Economia e delle Finanze | Monte dei Paschi di Siena |
| Ministero dell'Industria e del Commercio | Facile.it |
| Istituto per la Vigilanza delle Assicurazioni | MutuiOnline |
| Poste Italiane | Repubblica.it |
| soldionline.it | Sole 24 ore |
| supermoney guida alle carte di credito | |

all the cells contents; instead, each cell will be separated from the others by a "new line" in order to isolate different sentences for the contexts construction.

Beautiful Soup is a python library for parsing html documents. Unfortunately, we noticed that it did not work that well in our case. In some cases the problem was text duplication; in other cases, Beautiful Soup even fails to extract text. For this reason, we navigate the whole html tree making use of functions developed by ourselves, based on Beautiful Soup routines.

At this point, we end up with a collection of textual documents. All the html structure has been removed. This collection provides the technical corpus from which word2vec will extract the "technical language word embeddings".

### 3.3.2. From the corpus to the word embeddings dictionary

Once we have proper textual documents, both the external and the internal ones are concatenated with wrapping to produce a single file `our_words.txt`. If the chosen atomic element is the word and not the lemma, this is already our corpus, which is ready to be analysed by word2vec. This is typically the case, the reason being that our corpus is huge. If the corpus were not that large, we would need to go through lemmas. To understand why, think of a verb and its many conjugated forms: we could have many words ("scontato", "sconteranno", "sconta", ...) for the single lemma "scontare". If the corpus is not large enough each of them will have only a few occurences and the

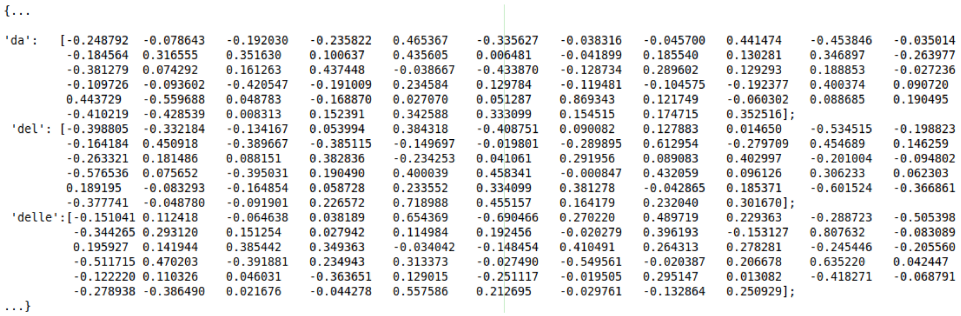



**Figure 3.3:** Example of some items of the word embeddings dictionary.

resulting world embedding would be not that significant. On the other end, it is clear than having all the words in place (instead of lemmas) we are going to end up with a plethora of word embeddings, which could make it difficult to handle all of them. All in all, the collection of all the word embeddings will be our dictionary: the richer the dictionary, the more effective we expect Soundlike will be and that's why we favour a large dictionary and we do not retain only lemmas.

word2vec will be run with the following features (see Sec. 3.2.1).

- We fix the dimension of word embeddings equal to 64.
- The context window size is 5 words.
- We set a threshold equals to $10^{-4}$ as sample parameter.
- We set the "negative label" parameter to 5.

Fed with the corpus `our_words.txt`, word2vec returns the `vec64.txt` file. This is not yet our dictionary: by further formatting the output, we construct the word embeddings dictionary as a collection of pairs (key, value). We have a key for every string of the corpus. The value associated to each key is the word embedding, i.e. a vector with 64 entries. An example of some items is in Fig. 3.3.

### 3.3.3. From the word embeddings dictionary to the document embeddings dictionary

Once the dictionary has been established, the corpus is no longer needed as a whole. We thus retain only the documents among which an answer for any question has to be selected. This is what we referred to as the internal bank documentation, all together are 16480 documents. As explained in Sec. 3.2.4, we need T2K to analyse them to recognise the POS (remember: we will retain adjectives, substantives, verbs). T2K processes the documents all

together, but keeping track of their identifications. This is easily done if we structure them with an header and a footer as follows

| | |
|---|---|
| header | `<doc id="document identifier" title="some title">` |
| | ... |
| | `textual contents` |
| | ... |
| footer | `</doc>` |

The identifier "`document id`" is the file name concatenated with its path. It will become the key of the document embeddings dictionary. The resulting POS-tagged file will have this structure:

| | |
|---|---|
| header | `<doc id="document identifier" title="some title">` |
| | index \| word \| lemma \| POS \| specific POS \| features |
| footer | `</doc>` |

where columns meanings are the same as explained in Sec. 3.2.2.

T2K leaves us with an output that is further processed to produce some sort of "intermediate" dictionary. As a dictionary, it is made of entries of the form (key,value), but it is not the document embeddings dictionary we are interested in. It is what we can call the "pre-hyper-dictionary" whose keys are the documents ids. The associated values are 3 different nested dictionaries for adjectives, substantives and verbs whose keys are "A", "S", "V". The associated values are in turn the collections of words which have been recognized as specific POS in the given document. If the $i$th document has $N_i$ adjectives, $M_i$ substantives and $K_i$ verbs, we have

$$\begin{aligned}
\texttt{preHyperDic} = \{\texttt{'id1'} : \quad & \texttt{'A'} : [a^{(1)}, a^{(2)}, ..., a^{(N1)}], \\
& \texttt{'S'} : [s^{(1)}, s^{(2)}, ..., s^{(M1)}], \\
& \texttt{'V'} : [v^{(1)}, v^{(2)}, ..., v^{(K1)}], \\
\texttt{'id2'} : \quad & \texttt{'A'} : [a^{(1)}, a^{(2)}, ..., a^{(N2)}], \\
& \texttt{'S'} : [s^{(1)}, s^{(2)}, ..., s^{(M2)}], \\
& \texttt{'V'} : [v^{(1)}, v^{(2)}, ..., v^{(K2)}], \\
& \ldots\}
\end{aligned}$$

**Example of a single entry of the pre-hyper-dictionary.**
Document id: soundlike_schedaprod_Conto private_0001.txt.
Textual content: "Che cosa è? Conto Private è un pacchetto composto da c/c e servizi bancari dedicati alla clientela Private (P1, P2 e P3) e ai clienti dei

| | | | | | | |
|---|---|---|---|---|---|---|
| 1 | 1 | Che | che | D | DQ | num:n gen:n |
| | 2 | cosa | cosa | S | S | num:s gen:f |
| | 3 | è | essere | V | V | num:s mod:i per:3 ten:p |
| | 4 | ? | ? | F | FS | |
| 2 | 1 | Conto | conto | S | S | num:s gen:m |
| | 2 | Private | Private | S | SP | |
| | 3 | è | essere | V | V | num:s mod:i per:3 ten:p |
| | 4 | un | uno | R | RI | num:s gen:m |
| | 5 | pacchetto | pacchetto | S | S | num:s gen:m |
| | 6 | composto | composto | A | A | num:s gen:m |

**Figure 3.4:** An example of POS-tagging.

CF. Il canone di conto corrente viene rimborsato in funzione della raccolta gestita del cliente. Il Conto Private è riservato solamente ed esclusivamente a clientela Private e ai clienti dei CF".
Resulting POS-tagged file: see Fig. 3.4.
The pre-hyper-dictionary entry related to this document, using word as atomic elements, is:

```
hyperDic["soundlike_schedaprod_Conto private_0001.txt"] =
        'A' : [bancari, corrente] ,
        'S' : [cosa, conto, pacchetto, servizi, clientela, ... ],
        'V' : [è, composto, viene, rimborsato, riservato];
```

The next step is replacing every string with its word embedding taken from the word embeddings dictionary. Having this structure, it is easy to implement a given representation of document embedding (e.g. by super vectors). Moreover, the calculation of similarities (given a definition) is fast.

As mentioned before, every question can be treated in the same way as bank documents. It will be represented by 3 hyper-vectors, one for its adjectives, one for its substantives and the last one for its verbs. The only difference is that we have to repeat the above procedure on the spot, rather than having an analysed file at hand.

## 3.4. A naive model evaluation

Before letting the customer evaluate the Soundlike performance, we set up a benchmark based on 6 questions for which we know the answers, i.e. the documents matching the questions. In the followings sections, the symbol "QA #" stands for a correct (question, answer) pair: this is what we call a gold standard. Notice that in some cases the correct answer is not unique: two or more different documents can be possible correct answers. This is ex-

actly the case of "QA5": 3 different documents are possible correct answers.

For each gold standard we performed 25 experiments, each for any combination of a given choice of the criterion used to construct hyper vectors (this is what we can call an intra-POS choice) and a given choice of the criterion used to compute document similarity (this is what we can call an extra-POS choice). The 25 combinations can be read according to the following table

| intra-POS choices | extra-POS choices |
|---|---|
| 1. mean | A. adjective hyper-vector similarity |
| 2. sum | B. substantives hyper-vector similarity |
| 3. max pooling | C. verbs hyper-vector similarity |
| 4. min pooling | D. mean on hyper-vector |
| 5. product | E. hyper-vectors concatenation |

We now have to keep in mind that we had two different word embeddings dictionaries: one was obtained using a "general language" corpus, the other using the "technical (i.e. bank) language" corpus. All in all, we repeated the 25 experiments twice, using the two different inputs. All the results are collected in tables 3.2 and 3.3.

## 3.5. First prototype: general word embedding

In a first approach, we don't train any word embeddings. We use the already trained word embeddings on corpus know in literature as itWaC [25]. It is a billion words corpus which has been obtained by crawling the .it domain on the web. The word embeddings are 128-sized. Since we have an already trained word embeddings dictionary, we must take the same conventions during the pre-processing of documents. If we don't, then we don't recognize the word embeddings of our words. For example, the itWaC notation lowercases all the strings starting with lower case character (e.g. "fAbiana" becomes "fabiana" , "fabiana" remains "fabiana"); if we keep "fAbiana" we won't find its word embedding components.

The results of each experiment are presented in Tab. 3.2. For each question there is a corresponding table of results. Rows and columns are to be interpreted as explained in Sec. 3.4 (see in particular the table over there).

So, how are the entries of the table obtained? Given a golden rule, any combination of intra-POS and extra-POS choices results in a list of similarity scores, one for each of the 16480 documents (i.e. one for each possible answer). These scores are shortlisted and in each entry of the table we can read the position of the correct answer in the relevant shortlist. Needless to say, the lower the number, the better the result; see for example entry (2,E)

for QA1: in this case the correct answer was indeed selected as Soundlike best option.

Actually we can notice that the choice (2, E) seems the best one for all the questions in the benchmark. It corresponds to sum component-wise the vectors in the same POS-category and to pass through the explicit super-vector notation for documents embeddings, in which an ordered concatenation of hyper-vectors is performed.

**Table 3.2:** Gold rule ranking using general purpose word embeddings.

| | | A | B | C | D | E |
|---|---|---|---|---|---|---|
| QA1 | 1 | 622 | 10 | 4874 | 99 | 160 |
| | 2 | 622 | 10 | 4874 | 99 | 1 |
| | 3 | 2824 | 3434 | 9762 | 4347 | 6199 |
| | 4 | 12016 | 3256 | 9099 | 9680 | 12062 |
| | 5 | 1982 | 15841 | 6673 | 13733 | 2289 |
| QA2 | 1 | 23 | 16 | 712 | 3 | 3 |
| | 2 | 23 | 16 | 712 | 3 | 3 |
| | 3 | 391 | 7838 | 4426 | 1246 | 1332 |
| | 4 | 2307 | 3727 | 3512 | 2075 | 2791 |
| | 5 | 10654 | 13494 | 1993 | 9560 | 11125 |
| QA3 | 1 | 234 | 5430 | 136 | 63 | 100 |
| | 2 | 234 | 5430 | 136 | 63 | 64 |
| | 3 | 5045 | 6535 | 265 | 1885 | 1295 |
| | 4 | 9050 | 7719 | 1156 | 4220 | 2758 |
| | 5 | 15141 | 7084 | 1687 | 11179 | 16132 |
| QA4 | 1 | 21 | 1 | 1727 | 7 | 28 |
| | 2 | 21 | 1 | 1727 | 7 | 1 |
| | 3 | 673 | 3 | 540 | 136 | 101 |
| | 4 | 838 | 1 | 738 | 182 | 172 |
| | 5 | 14477 | 931 | 2722 | 5277 | 15033 |
| QA5 | 1 | 5474 | 369 | 2673 | 548 | 560 |
| | 2 | 5474 | 369 | 2673 | 548 | 182 |
| | 3 | 5474 | 2519 | 5069 | 2922 | 3616 |
| | 4 | 5474 | 228 | 3020 | 741 | 2581 |
| | 5 | 5474 | 3461 | 1573 | 2487 | 612 |
| QA6 | 1 | 679 | 9 | 86 | 8 | 18 |
| | 2 | 679 | 9 | 86 | 8 | 8 |
| | 3 | 2235 | 13 | 2023 | 279 | 278 |
| | 4 | 3094 | 107 | 1493 | 510 | 512 |
| | 5 | 12786 | 162 | 15208 | 7621 | 14212 |

## 3.6. Second prototype: specific word embedding

As mentioned before, for the second prototype we collect word embeddings on a corpus based on bank documents (so the language is the technical one).

The results of each experiment is presented in Tab. 3.3 (same notation as in Sec. 3.4).

We can notice that, as in the previous case, the choice (2, E) appears to be the best one. In general, for this prototype we get better results. While this could be an accident, it is interesting to see how QA4 gets very good results for both prototypes.

**Table 3.3:** Gold rule ranking using specific language word embeddings.

| | | A | B | C | D | E |
|---|---|---|---|---|---|---|
| QA1 | 1 | 1182 | 12 | 4119 | 15 | 21 |
| | 2 | 1182 | 12 | 4119 | 15 | 1 |
| | 3 | 12845 | 1397 | 12825 | 9877 | 12334 |
| | 4 | 11993 | 12413 | 9579 | 12356 | 13194 |
| | 5 | 14680 | 14759 | 13863 | 14775 | 15672 |
| QA2 | 1 | 245 | 26 | 597 | 2 | 2 |
| | 2 | 245 | 26 | 597 | 2 | 13 |
| | 3 | 3725 | 8703 | 9271 | 6693 | 7624 |
| | 4 | 2546 | 4483 | 4308 | 2747 | 2785 |
| | 5 | 9784 | 1193 | 7121 | 3850 | 10582 |
| QA3 | 1 | 227 | 2574 | 18 | 27 | 34 |
| | 2 | 227 | 2574 | 18 | 27 | 44 |
| | 3 | 9218 | 8168 | 5052 | 6844 | 4691 |
| | 4 | 14021 | 6415 | 1534 | 8482 | 4953 |
| | 5 | 14303 | 1525 | 2448 | 8113 | 15580 |
| QA4 | 1 | 28 | 3 | 498 | 1 | 10 |
| | 2 | 28 | 3 | 498 | 1 | 1 |
| | 3 | 204 | 1 | 640 | 57 | 25 |
| | 4 | 40 | 2 | 178 | 35 | 21 |
| | 5 | 15141 | 2946 | 8329 | 8923 | 16134 |
| QA5 | 1 | 1811 | 126 | 4794 | 755 | 538 |
| | 2 | 1811 | 126 | 4794 | 755 | 70 |
| | 3 | 4338 | 649 | 1159 | 1147 | 2103 |
| | 4 | 4797 | 74 | 12298 | 5220 | 3066 |
| | 5 | 11404 | 1903 | 7432 | 4576 | 6693 |
| QA6 | 1 | 475 | 30 | 108 | 5 | 5 |
| | 2 | 475 | 30 | 108 | 5 | 7 |
| | 3 | 4638 | 305 | 2841 | 585 | 311 |
| | 4 | 720 | 200 | 2603 | 331 | 228 |
| | 5 | 2383 | 6537 | 7415 | 2888 | 3944 |

## 3.7. Observations and conclusions

Looking at the gold rules ranking in both situations, some results are not surprising. In particular, results for sum and mean (rows 1 and 2) are the

same for all columns but the last one. This does not come as a surprise: in the end, these two intra-POS choices only differ by a constant (i.e. the normalization of the mean). This is not the case for the last column, because this refers to sums computed on a larger vector (i.e. the super vector). We already made the observation that the best combination appears to be (2, E) (i.e. component-wise mean and super vector defined of hyper vectors). We notice that also choices (1, B), (1, D), (1, E), (2, B), (2, D) are good enough.

Given the huge amount of documents, it's really impressive to find the gold rules at the top of the shortlist and, in some cases, even in the first position.

When we don't find our gold rule in the first position, we look at the document which indeed is found in the first position. In many cases the answer seems to be a decent one, in others does not. What we have just noticed simply highlights a drawback of our procedure. Only for QA5 we provided more than one gold rule (actually only three). It could well be that in most cases there are many documents which decently match the question. Domain experts could help us with the construction of a better benchmark.

## 3.8. First steps towards future evolutions

Despite the good initial results, the project did not attract the client's interest because the internal AI infrastructure is not yet ready to accommodate a tool like Soundlike. In fact, they already use a chatbot as the software application to conduct an on-line chat conversation with customers. Since chatbot works with tagging, it could be better to have an auto-tagging tool. With this we mean a tool which automatically assigns the tags to a new document before it is deployed on the bank intranet, without requiring human assistance. This could enrich the chatbot knowledge without upsetting today's architecture and make employees life easier.

In the end we would like Soundlike to produce tagging more than similarity. We can think of 2 possible solutions.

- We could train a new machine learning algorithm to map the document embedding to a finite numbers of tags decided in advance. This is nothing but a classical classification task, i.e. a supervised algorithm for which a set of historical documents with their already assigned tags is needed.
- We could compute the similarity between the document embedding and either the word embeddings of the entire corpus dictionary or the

word embeddings of the words contained in the document itself. With this in mind we can select the tags in two possible ways. Given a similarity threshold, a word will be a tag every time its similarity is higher than the threshold. Given a number $k$ and the shortlist of word similarities we select as tags the first $k$ words. In the first case, the drawback could be ending up with cases in which either we find no tag or we find too many. In the second case an obvious drawback is that we have no control on how good the similarity score is for selected tags.

# Final overview

This work is the result of the collaboration between the academic and the business word. It is a virtuous example of how different realities can complement and enhance each other. The state of the art reached at universities, like the University of Parma, or at public research institutions like the ItaliaNLP Lab of CNR can lead to a commercial gain when it meets a vibrant and dynamic enterprise setting, like at Energee3.

The presented solutions are just a starting point since many remarkable results and interesting applications may be obtained. In particular, the learning protocol we developed for phase transitions may be successfully also in other fields. Since it can be used every time a pattern recognition task is required, it may find applications in the business area as well.

# Bibliography


[1] Y. B. I. Goodfellow and A. Courville, *Deep Learning*. The MIT Press, 2016.

[2] D. Barber, *Bayesian Reasoning and Machine Learning*. Cambridge University Press, 2012.

[3] K. Murphy, *Machine Learning: A Probabilistic Perspective*. The MIT Press, 2012.

[4] R. T. T. Hastie and J. Friedman, *The Elements of Statistical Learning*. Springer Nature, 2013.

[5] A. Müller and S. Guido, *Introduction to Machine Learning with Python: A Guide for Data Scientists*. O'Reilly Media, 2016.

[6] Y. Abu-Mostafa, "Learning from data - Machine Learning course recorded at a live broadcast from Caltech."

[7] A. Ng, "Machine Learning on Coursera."

[8] V. Nair and G. Hinton, "Rectified Linear Units Improve Restricted Boltzmann Machines," pp. 807–814, 2010.

[9] N. Goldenfeld, *Lectures on Phase Transitions and the Renormalization Group*. Addison Wesley, 6 1972.

[10] L. Onsager, "Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition," *Phys. Rev.*, vol. 65, pp. 117–149, Feb 1944.

[11] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller, "Equation of state calculations by fast computing machines," *The Journal of Chemical Physics*, vol. 21, no. 6, pp. 1087–1092, 1953.

[12] A. Sokal, *Monte Carlo Methods in Statistical Mechanics: Foundations and New Algorithms*. Springer, Boston, MA, 1997.

[13] R. Swendsen and J.-S. Wang, "Nonuniversal critical dynamics in Monte Carlo simulations," *Phys. Rev. Lett.*, vol. 58, pp. 86–88, Jan 1987.

[14] P. Ponte and R. Melko, "Kernel methods for interpretable machine learning of order parameters," *Phys. Rev. B*, vol. 96, p. 205146, Nov 2017.

[15] B. L. C. Giannetti and D. Vadacchino, "Machine Learning as a universal tool for quantitative investigations of phase transitions," *Nuclear Physics B*, vol. 944, p. 114639, 2019.

[16] S. Wetzel and M. Scherzer, "Machine learning of explicit order parameters: From the Ising model to SU(2) lattice gauge theory," *Phys. Rev. B*, vol. 96, p. 184410, Nov 2017.

[17] T. G. A. K. G. Cossu, L. Del Debbio and M. Wilson, "Machine learning determination of dynamical parameters: The Ising model case," *Phys. Rev. B*, vol. 100, p. 064304, Aug 2019.

[18] J. Carrasquilla and R. Melko, "Machine learning phases of matter," *Nature Physics*, vol. 13, p. 431–434, Feb 2017.

[19] L. van der Maaten and G. Hinton, "Visualizing Data using t-SNE," *Journal of Machine Learning Research*, vol. 9, no. 86, pp. 2579–2605, 2008.

[20] G. A. D. Bachtis and B. Lucini, "Extending machine learning classification capabilities with histogram reweighting," *Physical Review E*, vol. 102, Sep 2020.

[21] "In preparation, to be issued soon,"

[22] "http://www.italianlp.it/."

[23] Z. Harris, "Distributional Structure," *WORD*, vol. 10, no. 2-3, pp. 146–162, 1954.

[24] A. Lenci, "Semantica distribuzionale. Un modello computazionale del significato," *COMPTER PARLER SOIGNER - Tra linguistica e intelligenza artificiale*, pp. 39–53, 2016.

[25] A. F. M. Baroni, S. Bernardini and E. Zanchetta, "The WaCky wide web: a collection of very large linguistically processed web-crawled corpora," *Language Resources and Evaluation*, vol. 43, pp. 209 – 226, 2009.

[26] P. Turney and P. Pantel, "From Frequency to Meaning: Vector Space Models of Semantics," *Journal of Artificial Intelligence Research*, vol. 37, pp. 141–188, 2010.

[27] G. D. M. Baroni and G. Kruszewski, "Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic

vectors," *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics*, vol. 1 - Long Papers, pp. 238–247, 2014.

[28] G. C. T. Mikolov, K. Chen and J. Dean, "Efficient Estimation of Word Representations in Vector Space," 2013.

[29] K. C. G. C. T. Mikolov, I. Sutskever and J. Dean, "Distributed Representations of Words and Phrases and their Compositionality," 2013.

[30] R. Petrolito and F. Dell'Orletta, "Word Embeddings in Sentiment Analysis," in *CLiC-it*, 2018.

[31] A. C. F. Dell'Orletta, G. Venturi and S. Montemagni, "T2K^2: a system for automatically extracting and organizing knowledge from texts," in *Proceedings of the Ninth International Conference on Language Resources and Evaluation (REC'14)*, (Reykjavik, Iceland), pp. 2062–2070, European Language Resources Association (ELRA), may 2014.