Machine Learning in a bidimensional ferromagnetic Ising model: predictions on lattices temperatures and phase transitions

Sabina Gaia Tomasicchio, Simone Rocca, Ilham Dekhissi
(Dated: February 14, 2024)

A theoretical overview of the Ising model for ferromagnetism is presented, with a focus on the bidimensional case without the presence of an external magnetic field. Subsequently, a description of a selection of regression models, i.e. least-square regression, Ridge and Lasso Regression is given. These are used to predict the temperature of each lattice configuration of spins. This regression task is also performed by invoking Neural Networks, in particular both a Feed-forward Neural Network (FFNN) and a Convolutional Neural Network (CNN) are implemented and trained. The Mean Squared Error (MSE) and the R^2 metrics are computed in order to compare the different models. From that, we conclude that Ridge gives the best "relative" results among the regression models. However, the best "absolute" results are given, as expected, by the CNN. A CNN is also used to perform a classification task on the lattices, to classify them as being above or below the critical temperature T_c. This task is carried out successfully with an accuracy of 98.9%. Lastly, through logistic regression we are able to estimate the critical temperature as the value for which the probability $p(T > T_c) = 0.5$, getting $T_c = 2.273$ J/k_B that, compared to the theoretical one $T_c = 2.269$ J/k_B [15] presents a relative error of the 0.17%. The 6000 data samples used throughout the entire work have been generated thanks to a Monte Carlo Markov Chain algorithm (Metropolis).

I. INTRODUCTION

At its core, Physics aims to understand the underlying principles that govern the Universe and unravel its phenomena. Machine Learning (ML) techniques, allowing for gathering insights from data and making accurate predictions, have emerged as transformative tools in scientific research. These methods are enabling researchers to analyze vast datasets, identify patterns, and generate hypotheses, facilitating a deeper understanding of complex systems. Moreover, the versatility of ML extends beyond the realm of the natural sciences, finding applications across a wide spectrum of disciplines.

Recently, ML methods have been employed to tackle statistical physics problems and have shown promising predictions. Given their nature, a valid playground to test these techniques is represented by lattice models, and the Ising model is by far the most intensively studied one. These models have a variable at each site of a regular grid and a Hamiltonian or evolution law for these variables. The Ising model has been extensively used for solving a variety of problems: besides the phenomenon of ferromagnetism, it has been applied, for instance, to the lattice gas, the spin glasses and the activity of neurons in the brain. In the context of ferromagnetism, it is useful to model phase transitions. Ferromagnetic materials exhibit a long-range ordering phenomenon at the atomic level which causes the unpaired electron spins to line up parallel with each other in a region called a domain. For a given ferromagnetic material, the long range order abruptly disappears at a certain temperature which is called the Curie temperature for the material, that is the critical temperature T_c where the phase transition takes place [17].

In particular, predicting the phase diagrams of interacting spin systems using ML has become an area of research interest [1]. It has been also shown that it is possible to perform classification tasks on the Ising model with reasonable accuracies [11].

The focus of this project is supervised learning, which consists of mainly regression models, classification models and Neural Networks (NNs), whose name was inspired by neuroscience since they are made of artificial "neurons". In supervised learning we feed input data and labels which represent the output data into the system. The better the labels fit to the output of the network the more the weights are changed towards these values. The goal is to apply these models to the Ising model. We perform the regression task of predicting the temperature of a given lattice spin configuration and the classification task of determining whether these lattices are above or below the critical temperature T_c.

There are many different kinds of neural networks: we have developed and trained Feed-forward Neural Networks (FFNNs) and Convolutional Neural Networks (CNNs). FFNNs are among the simplest and most commonly used networks in various Machine Learning tasks, including classification, regression, and pattern recognition. The goal of a feedforward network is to approximate some function f^*, i.e. $y = f^*(x)$. They are called feed-forward because information flows through the function being evaluated from x, through the intermediate computations used to define f and finally to the output y. There are no feedback connections in which outputs of the model are fed back into itself [5]. Another particular type of NNs, employed especially in image processing are CNNs, which are a specialized kind of neural network for processing data that has a known grid-like topology [5]. Nowadays, they are also used to identify phases of spin models [2].

The work is organized as follows. We open, in Section I, with an introduction that covers the motivations of this work. In Subsection I A, a brief overview of the Ising model, subject of study, is carried on. After that, we commence in Section II with a description of the methods used, in Subsection II A, II B and II C we cover the theory behind the regression models, FFNNs and CNNs respectively. In Subsection II D, a succinct explanation of how the data have been generated, together with the Metropolis algorithm, is given. Then, we turn in Section III to the presentation and discussion of the results obtained. In Subsection III A, we apply different regression models, i.e. least-squares regression (`LinearRegression` from the `sklearn` package), Ridge regression and Lasso regression, on the study of the temperature. For these methods we also perform parameter search using cross validation. To check the robustness of such models, we also try a bootstrapping approach. In Subsection III B, we present the results of the regression obtained with the use of a FFNN. In Subsection III C, we investigate the same regression task but using a CNN. A CNN is also implemented for a classification task on the temperature dependent phase transition; the corresponding analysis of the results is conducted in Subsection III D. A logistic regression is then performed to estimate with precision the critical temperature value T_c to be compared to the analytical one found by L. Onsager [15]. Section IV concludes our work, where a small outlook of our results as well as possible improvements to be performed in the future are given. Finally, Appendix A carries some supplementary plots regarding Section III.

The code at the base of the experiments reported can be found at the following GitLab page.

A. Theory

In this project, our system of interest will be a two dimensional (2D) Ising model that we will use to describe a magnet. The model postulates a lattice with a magnetic dipole or spin on each site. We will consider a square lattice of length L and N sites such that $N = L^2$, with a spin s_i on each site. A spin configuration refers to the spin state of the entire system and is denoted as $\mathbf{s} = (s_1, s_2, ..., s_\text{N})$. Each individual spin has only two possible states: "up" $s_\text{i} = +1$ and "down" $s_\text{i} = -1$. Therefore, there are a total of 2^N possible configurations of the system, which are specified by the orientations of the spins on all N sites. The energy of any particular state is given by the Ising Hamiltonian

$$E(\mathbf{s}) = -J \sum_{\langle kl \rangle}^{N} s_\text{k} s_\text{l} - \mu H \sum_{k}^{N} s_\text{k}, \tag{1}$$

where $\langle kl \rangle$ denotes that the sum is to be carried out over nearest-neighbor pairs of lattice sites, μ is the magnetic moment, H is the external and spatially constant magnetic field and J is the coupling constant. If J is positive the interaction is ferromagnetic, if J is negative the interaction is anti-ferromagnetic. In our case we will consider $J > 0$ and $H = 0$, therefore the total energy of the system becomes

$$E(\mathbf{s}) = -J \sum_{\langle kl \rangle}^{N} s_\text{k} s_\text{l}. \tag{2}$$

We notice that the energy is lowest when all spins are aligned, either up or down.

Furthermore, we apply at this model the so-called periodic boundary conditions, which define a configuration where each spin has always four neighbours and those at the edges of the lattice interact with the spins at the geometric opposite edges. Therefore, the configuration will be the surface of a three dimensional torus. Periodic boundary conditions ensure that all spins have the same number of neighbours and local geometry, and that there are no special edge spins which have different properties from the others; all are equivalent and the system is completely translationally invariant [14].

Regarding the units used, we will consider the spins to be unitless, this means that J has units of energy. Therefore the temperatures will be expressed in J/k_B.

Given a system with temperature T, the probability for the system state $\mathbf{s}$ is given by the Boltzmann distribution:

$$w(\mathbf{s}; T) = \frac{1}{Z} e^{-\beta E(\mathbf{s})}, \tag{3}$$

where Z is the partition function defined as

$$Z = \sum_{\text{all possible } \mathbf{s}} e^{-\beta E(\mathbf{s})}. \tag{4}$$

and $\beta = 1/k_B T$ is the "inverse temperature", with $k_B \approx 1.38 \cdot 10^{-23}\ J \cdot \text{K}^{-1}$ being the Boltzmann constant. Let us make some considerations about the temperature dependent behaviour of our system. At $T = 0$, it will be in its ground state, with all spins pointing in the same direction, tending towards a state with the highest possible degree of order. At $T = \infty$, the entropy dominates. The spins will be randomly oriented and energy will be maximized. The two regimes are separated by a phase transition that occurs at the critical temperature T_c (Curie temperature). During this phase transition there is a sudden change from an ordered phase to a disordered phase as the temperature is increased [16]. Such a behavior is an example of what are called critical phenomena, that are normally marked by one or more thermodynamical variables which vanish above a critical point. The critical temperature T_c for an infinite 2D Ising model, found analytically by L. Onsager [15], is

$$T_c(L = \infty) = \frac{2}{\ln\left(1 + \sqrt{2}\right)} J/k_B \approx 2.269\ J/k_B. \tag{5}$$

In the region close to T_c, which is called the critical region, the system tends to form large clusters of predominantly up or down pointing spins, as we approach the critical temperature from above.

II. METHODS

In the present Section, we aim to give a clear and concise description of each of the methods chosen to perform the study. We start with the regression methods in Subsection II A and then we move to Subsections II B and II C for an explanation of the FFNN and the CNN. At the end, Subsection II D gives some insights on how the data have been generated.

A. Regression methods

The content of this Subsection is widely based on [8].

1. *Ordinary Least Squares (OLS)*

The primary objective of the Ordinary Least Squares (OLS) method is to establish a relationship between a random variable, denoted as y, and a set of variables $\boldsymbol{x} = [x_0, x_1, \dots, x_{n-1}]^T$ within a given dataset. Initially, the nature of the relationship between the response variable $\boldsymbol{y} = [y_0, y_1, \dots, y_{n-1}]^T$ and the variables in $\boldsymbol{x}$ is unknown. The first step involves creating a matrix $\boldsymbol{X}$, referred to as the design matrix, which encompasses the variables present in the dataset.

Subsequently, a linear relationship is assumed between the response variable $\boldsymbol{y}$ and the variables in $\boldsymbol{x}$, denoted as $y_i = f(X_{i,*})$. If this relationship takes the form of a polynomial of degree $n-1$, the equation describing the data can be expressed as follows

$$y_i(x_i) = \tilde{y} + \epsilon_i = \sum_{j=0}^{n-1} \beta_j x_i^j + \epsilon_i. \tag{6}$$

The symbols β_j denote the fitting parameters that require determination, while the ϵ_i characterize the model's errors in the approximation. The relationship can be alternatively expressed as

$$\boldsymbol{y} = \boldsymbol{X}\boldsymbol{\beta} + \boldsymbol{\epsilon}, \tag{7}$$

where $\boldsymbol{\beta} = [\beta_0, \beta_1, \dots, \beta_{n-1}]^T$ and $\boldsymbol{\epsilon} = [\epsilon_0, \epsilon_1, \dots, \epsilon_{n-1}]^T$. To determine the optimal parameters β, we initially consider an approximation given by

$$\tilde{\boldsymbol{y}} = \boldsymbol{X}\boldsymbol{\beta}, \tag{8}$$

we then introduce a function, called the cost function, that quantifies the discrepancy between the exact values y_i and the parameterized ones $\tilde{y}_i$

$$C(\boldsymbol{\beta}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2 = \frac{1}{n}\left\{(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta})\right\}. \tag{9}$$

Minimizing this function is possible in order to get the best value of $\boldsymbol{\beta}$

$$\boldsymbol{\beta} = (\boldsymbol{X}^T\boldsymbol{X})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{10}$$

2. *Ridge Regression*

Ridge Regression is a regularization technique used in linear regression to address the issue of multicollinearity, where independent variables are strongly correlated with each other. This correlation can lead to unstable coefficients that are highly dependent on the specific training data, making the model very sensitive to small variations in input data. Ridge Regression introduces a regularization term into the model's objective function to limit the magnitude of the coefficients. The cost function for Ridge regression is

$$C(\boldsymbol{X}, \boldsymbol{\beta}) = \frac{1}{n}||\boldsymbol{y} - \boldsymbol{X}\boldsymbol{\beta}||_2^2 + \lambda||\boldsymbol{\beta}||_1, \tag{11}$$

where $\boldsymbol{X}$ is the design matrix and λ is a hyperparameter. Moreover, we require that $||\boldsymbol{\beta}||_2^2 \leq t$, where t is a finite number larger than zero. By minimizing the function, we obtain a set of parameters β that can be expressed as follow

$$\hat{\boldsymbol{\beta}}_{\text{Ridge}} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}\boldsymbol{X}^T\boldsymbol{y}. \tag{12}$$

3. *Lasso Regression*

Lasso (short for Least Absolute Shrinkage and Selection Operator) Regression is another regularization technique used in linear regression, particularly when dealing with multicollinearity and variable selection. Like Ridge Regression, Lasso adds a regularization term to the objective function to control the magnitude of the coefficients.

The cost function for Lasso regression is

$$C(\boldsymbol{\beta}) = \sum_{i=0}^{p-1}(y_i - \beta_i)^2 + \lambda\sum_{i=0}^{p-1}|\beta_i|. \tag{13}$$

Unlike Ridge, Lasso has the unique property of driving some coefficients exactly to zero, effectively performing variable selection. This sparsity-inducing property makes Lasso useful when one wants to identify and exclude irrelevant features from the model. Assuming that the design matrix $\boldsymbol{X}$ is given by unit matrix, that is a square diagonal matrix with ones only along the diagonal. In this case we have an equal number of rows and columns $n = p$. By minimizing the cost function, we get

$$\hat{\beta}_i^{\text{Lasso}} = \begin{cases} y_i - \frac{\lambda}{2} & \text{if} \quad y_i > \frac{\lambda}{2} \\ y_i + \frac{\lambda}{2} & \text{if} \quad y_i < -\frac{\lambda}{2} \\ 0 & \text{if} \quad ||y_i|| \leq \frac{\lambda}{2} \end{cases}. \tag{14}$$

4. *Resampling techniques*

Let us take the example of a dataset that has a limited number of data points, in this case obtaining accurate estimates like the Mean Squared Error or the variance can be challenging. In such situations, resampling methods offer a valuable approach to approximate these estimates by rearranging the data. However, with a balanced and large dataset it appears that these resampling methods may not significantly enhance our comprehension of the model.

Bootstrapping is a resampling technique used to estimate statistical properties of a dataset, particularly in the context of training data. In this method, a sample dataset is drawn - with replacement - from the original dataset n times. At each iteration, an estimate is obtained from the resampled dataset, contributing to the inference of the overall estimate for the dataset. The bootstrap method provides a robust way to assess the variability and distribution of estimates by repeatedly sampling from the dataset, making it a valuable tool in statistical analysis and model evaluation.

With cross-validation, the dataset is partitioned into subsets known as $k-$folds. Each of these subsets contains nearly equal data points and forms exhaustive, mutually exclusive partitions. In this context, a method such as OLS, Ridge, etc., is employed k times. During each iteration, one of the $k-$folds serves as the test dataset, while the remaining $k-1$ folds collectively function as the training dataset. This process ensures that each subset acts as a test set exactly once, avoiding non-uniform resampling and providing a comprehensive evaluation of the model across different data partitions.

In this project, both bootstrapping and cross-validation have been taken into account.

5. *MSE and R^2 metrics*

In assessing the effectiveness of the predictive model for the data, two distinct metrics were employed: the Mean Squared Error (MSE) and the R^2 value. The MSE serves as a metric to calculate the risk corresponding to the expected value of the squared error, its definition is given by

$$\mathrm{MSE}(\boldsymbol{y},\tilde{\boldsymbol{y}}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2, \tag{15}$$

where $\tilde{y}_i$ is the predicted value of the $i-th$ sample and y_i is the true value. Ideally, we would prefer this function to be equal to zero. If achieving zero is not possible, a value close to zero would indicate that the fit we are performing serves as a good representation of our data.

Another function introduced for calculating the coefficient of determination is the R^2 score, defined as

$$R^2(\boldsymbol{y},\tilde{\boldsymbol{y}}) = 1-\frac{\sum_{i=0}^{n-1}(y_i-\tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i-\bar{y}_i)^2}, \tag{16}$$

where the mean value of $\boldsymbol{y}$ has been defined as

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i. \tag{17}$$

The model represents perfectly the data when the R^2 value is equal to 1.

B. FFNN

In the present Subsection, we will define what Feedforward Neural Networks (FFNNs) are and discuss the reasons for their use. Additionally, we will describe the architecture of the FFNN implemented for the regression task.

FFNNs are the first and simplest type of artificial NNs that were devised. They are typically represented by composing together many different functions. Each function, which represents a layer, receives inputs and calculates an output after having evaluated the internal weights. The weights are randomly changed, when a changed weight results in a better recognition its value will be updated. The number of layers gives the depth of the model, hence the name "deep learning". The final layer of a feedforward network is the output one. The intermediate layers are called "hidden". The dimensionality of these hidden layers determines the width of the model [5]. FFNNs are also called Deep Feedforward Networks, which differ from "normal" NNs for the number of hidden layers as well as the number of neurons per layer.

Within the architecture of a FFNN, the equation

$$y = f\left(\sum_{i=1}^{n} w_i x_i\right) = f(u), \tag{18}$$

captures the essence of a single neuron's operation, placing emphasis on the weighted summation of inputs. It establishes a relationship between the weighted inputs x_i, where the weights w_i determine the significance of each input in the final output, and the output y.

A fundamental ingredient of every NN is the so called activation function, represented by f in eq.(18). Without this function, the network will be limited to linear transformations. Therefore, the activation function plays a crucial role in enabling neural networks to manage non-linearities in the data. Different choices of non-linearities lead to different computational and training properties for neurons. The most used activation function is a rectified linear unit or ReLU [4], defined as $\mathrm{ReLU}(a) = \max(a, 0) = a\mathbb{I}(a > 0)$. Essentially, this function operates like a switch. For a positive input a, the output equals the input, and when the input is negative, the output becomes zero [13].

By maintaining a non saturating slope for large inputs, the ReLU avoids the vanishing gradient problem facilitating a faster convergence of the model. This problem occurs when the gradient of the loss function with respect to the weights become extremely small during the optimization of the network through the layers [20]. In contrast, activation functions like tanh [18] restricts its output values to a specific range, specifically mapping its input values to fall within the interval $[-1, 1]$. This limitation imposed by the tanh on its output range can lead to saturation effects, causing gradients to approach zero during the optimization of the NN. This characteristic of the hyperbolic tangent activation function contributes to the vanishing gradient issue in deep networks. This feature makes ReLU a favored choice.

Another important regularization schemed that has been widely adopted is Dropout [19]. The basic idea of Dropout is to prevent overfitting by reducing spurious correlations between neurons within the network by introducing a randomization procedure [11]. With overfitting we mean the phenomenon for which models are tricked into thinking that statistical noise encodes information, it usually leads to a better loss function in the train set compared to the test (or validation) one.

Lastly, regarding the optimizer to use for the training process, our choice fell on the largely used and popular Adam (short for Adaptive Moment Estimation) [9]. It combines the advantages of other popular optimization algorithms such as AdaGrad [3] and RMSProp [21]. At its core, Adam maintains two moving averages of gradients: the first moment (the mean) and the second moment (the uncentered variance). These moments are computed exponentially, with bias correction to account for their initialization at zero. During training, Adam updates the parameters of the neural network by adjusting them in the direction of the gradients. It does so by computing an adaptive learning rate for each parameter based on the estimated first and second moments of the gradients. This adaptive learning rate allows Adam to handle sparse gradients and noisy data more effectively, leading to faster convergence and improved performance compared to traditional optimization methods. Overall, Adam's combination of adaptive learning rates and momentum makes it a powerful and by far the most widely-used optimizer for training neural networks.

The FFNN implemented and trained in this case is articulated as follows. It starts with a linear layer, with 1600 input neurons and N output neurons, at which the ReLU activation function is applied. After that we have a dropout regularization layer with value of 0.2. At the end, we have a linear layer with N input neurons and one single output neuron. The number N can take one of the values $128, 256, 512, 1024, 2048$, for reasons that will be clearer when we discuss the results in Section III C. In Table I we store all the corresponding hyperparameters used.

Hyperparameter	Value
Learning rate	10^{-4}
Epochs	10^{2}
Optimisation	Adam
Batch size	64
Hidden layers	1
Layer size	2048
Dropout	0.2

TABLE I. Overview of hyperparameters of the best FFNN implemented.

C. CNN

In this subsection, we are going to explain what Convolutional Neural Networks (CNNs) are and the motivations behind their use. Moreover, we are going to illustrate the CNN's architecture implemented for both the regression and the classification tasks. They carry the same structure with the exception of a sigmoid activation function applied to the output for the classification case.

CNNs [10] are similar to FFNNs in a sense that they are made up of neurons that have learnable weights and biases, with the addition that are translationally invariant and respect locality of the input data [11]. At the basis of each CNN there is the convolution operation, defined by a mathematical operation on two function in order to produce a third function that expresses how the shape of one gets modified by the other. Let us define a continuous function [5] given by

$$y(t) = \int x(a)w(t-a)\mathrm{d}a, \tag{19}$$

where $x(a)$ represents a so-called input and $w(t-a)$ is normally called the weight function or kernel. Written in a more compact form, eq.(19) becomes

$$y(t) = (x * w)(t), \tag{20}$$

where the operation denoted with an asterisk is called convolution.

For example, if we have a two-dimensional image I as input, we can have a filter defined by a two-dimensional kernel K. This leads to an output S defined as

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n), \tag{21}$$

where we have also operated a discretization.

Notice the convoluted lattice is smaller than the original one. This downscale is responsible for the dimensionality reduction in the learnable parameters. This downscaling ratio is defined by the filter sizes used and also the stride. The stride defines how much the applied filter moves to the right after applying the convolution operation in different patches of the image; it is the step the filter takes between patches.

A typical layer of a CNN consists of three stages. In the first one, the layer performs several convolutions in parallel to produce a set of linear activations. In the second one, each linear activation is run through a nonlinear activation function. In the third stage, a pooling function is used to modify the output of the layer further [5].

A pooling function replaces the output of the net at a certain location with a summary statistic of the nearby outputs. There are different types of pooling techniques. In our case, we use the "max pooling" [22] which reports the maximum output within a rectangular neighborhood.

As part of "data preprocessing" a padding operation is performed, meaning that the edges of the lattice grid are filled with a specific value. Usually, fully connected layers are added at the end of the CNN in order to extract more effectively the patterns that were chosen and kept throughout the convolutional and pooling layer feature extractions. Finally, by connecting the output of the dense layers to one or more than one output neuron, we are able to put to use our CNN with a classification or regression task.

The CNN that we have implemented for this project is articulated as follows. It starts with a first 2D convolutional layer with one input channel and 16 output channels, using a 3×3 kernel with padding and stride equal to 1. Next, we have a second 2D convolutional layer with with 16 input channels, 32 output channels, and the same values for kernel, padding and stride as the previous one. After a dropout layer with a value equal to 0.1, we have a first fully connected layer with an input size of 800 and 128 output neurons; a second fully connected layer with 128 input neurons and one output neuron ends the CNN. The 2D convolutional layers are each followed by a max pool layer of kernel size 2×2. The output of the max pooling layer is then flattened and fed to two dense layers. For the first dense layer, as well as the 2D convolutional layers, we employed the ReLU activation function.

In Table II, we collect all the hyperparameters that characterize the CNN implemented.

Hyperparameter	Value
Learning rate	10^{-4}
Epochs	10^2
Optimisation	Adam
Dense layers	2
Dense layers size	128
Dropout	0.1

TABLE II. Overview of hyperparameters of the CNN implemented.

D. Data generation

Let us now briefly discuss how we generated the data used for the entire project. The method implemented is a type of Markov Chain Monte Carlo (MCMC) algorithm known as Metropolis [7, 12].

We start by considering a square lattice of size $L = 20$, where each spin value is randomly selected as $+1$ or -1. The algorithm consists in iterating throughout the entire lattice and eventually "flipping" each spin, i.e. change its current value from ± 1 to ∓ 1.

When operating with regression models the data have been split in train and test assuming a $80 : 20$ proportion, whereas the validation set is taken care of by the cross validation. Concerning the parts where we employ NNs, the split between train, validation and test has been done following a $60 : 20 : 20$ proportion. Moreover, we have considered temperatures in the range $T \in [1.0, 4.0]$ J/k_B with a step of 0.01 J/k_B. Following what has been done in [6], to guarantee a balanced data set, each temperature value occurs 4 times. In total we produce, after 10^4 Monte Carlo cycles, 6000 samples. In Figure 1, we show three examples of the lattices' spin configurations for different temperatures values.

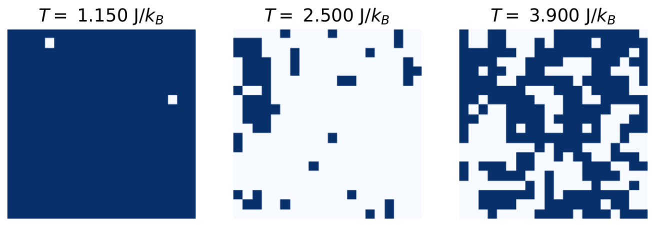

FIG. 1. Configurations of spins after 10^4 MC cycles for a $L = 20$ lattice at temperatures $T = 1.15, 2.50$ and 3.90 J/k_B. Dark blue indicates $s = +1$, light blue $s = -1$.

The problematic part of an actual Monte Carlo simulation resides in the appropriate selection of random states, according to the probability distribution (PDF) at hand. In order to generate new random states, Markov processes are used. A Markov process is a random walk with a selected probability for making a move, so the probability of each event depends only on the state attained in the previous event.

The acceptance rate of this spin change is modeled by a probability that depends on the system's temperature. The probability of this system's state is given by eq.(3), however we do not know the acceptance probability. If we move to a state with a lower energy, we always accept this move while if the energy is higher, we need to check the acceptance probability with the ratio between the probabilities from our PDF. This ratio is equal to $\exp(-\beta\Delta E)$ and it is compared with a random number: if it is greater than a given random number we accept the move.

Since we start with a random initial state, running the simulation for a high enough number of cycles is necessary to reach an equilibrium distribution and thus a statistically stable state of the system. That is why this procedure is repeated for 10^4 Monte Carlo steps.

The Metropolis algorithm is reported in algorithm 1, where matrix($\mathbf{s}$) is the matrix that represents our spin configuration, RN is a random number uniformly distributed between 0 and 1 and Bf is the Boltzmann factor $Bf = \exp(-\Delta E/T)$, where $\Delta E = (E_{\text{final}} - E_{\text{initial}})$ is the energy shift due to flipping a single spin. One so-called Monte Carlo cycle is shown, that corresponds to $N = L^2$

attempted spin flips, where N is the number of spins in the lattice $L \times L$.

Algorithm 1 Metropolis

```
**procedure** METROPOLIS(matrix($\mathbf{s}$), $Bf$)
    $L \leftarrow$ size(matrix($\mathbf{s}$))                ▷ Store size of the matrix
    **for** $i = 1, 2, ..., L^2$ **do**
        $x, y \leftarrow$ random                ▷ Pick random lattice site
        $\Delta E \leftarrow 2s_{xy}(s_{(x+1)y} + s_{(x-1)y} + s_{x(y+1)} + s_{x(y-1)})$ ▷ Compute energy difference
        **if** $RN \leq Bf(\Delta E)$ **then**
            $s_{xy} \leftarrow -s_{xy}$                ▷ Flip the spin
```

III. RESULTS AND DISCUSSION

In the following Section, we are going to present and discuss the results of our work. For better clarity, we have divided them into subsections. In Subsection III A, we are going to discuss the results obtained through the Regression methods chosen. In Subsection III B, the results concerning the use of a simple FFNN are presented. In Subsection III C, we show the results obtained from the regression task with a CNN; while in Subsection III D, we discuss the use of the CNN to perform the classification task.

A. Part I: Regression methods

The regression task consisted of predicting the temperature of the lattice configurations of spins. Here we are going to present the result obtained with the use of regression methods.

In Figure 2, we present the predicted and the true values of the temperature for each spin configuration of the test set, obtained through the `LinearRegression` framework from `sklearn`, which is based on the OLS method.

We then turn to Ridge Regression, in particular we first operate the search for the regularization parameter λ, using 5 folds cross-validation to assess the best parameter. To this regard in Figure 15 the mean R^2 over the different folds is represented as a function of λ. The search is performed over a space of 10 possible values logarithmically spaced between 10^{-4} and 10^5. The λ for which the R^2 is maximized is 10^4. In Figure 3, we show the plot of the predicted and the true temperatures for each of the test set's samples.

Concerning Lasso Regression we perform again the λ parameter search, similarly to what we did for Ridge Regression, that we display in Figure 16. In Figure 4, we plot the predicted and the true temperatures values for each lattice configuration of the test test, after having fixed λ to its optimal value $\lambda = 10^{-2}$.

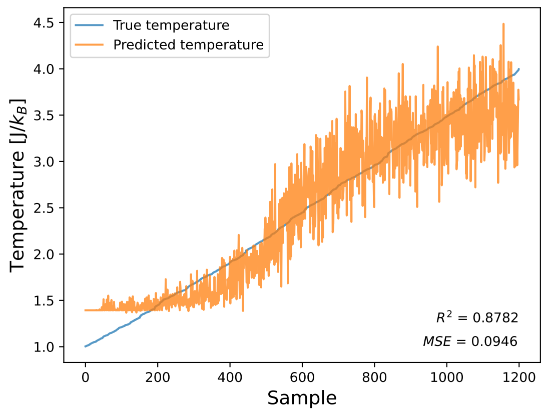

FIG. 2. Prediction over the test set of the temperatures of the given lattice configurations. The predictions were made with `LinearRegression` from `sklearn`. The values of the MSE and R^2 are also indicated.

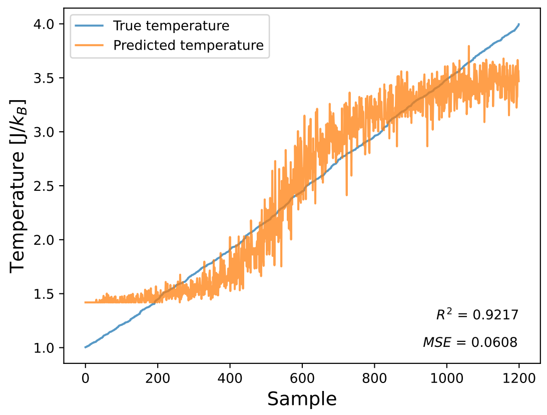

FIG. 3. Prediction over the test set of the temperature given lattice configurations. The predictions were made with Ridge regression and regularization parameter $\lambda = 10^4$. The values of the MSE and R^2 are also indicated.

We now investigate the use of bootstrapping on all the regression models already used so far, in order to verify the robustness of the predictions. More specifically, we fit each model on multiple bootstrapped samples from the training set. After that, we predict the temperatures on the test set and average them over all the bootstraps. The number of bootstraps chosen is, due to computational reasons, 100.

In Figure 5, we show the plot of the average predicted temperatures and the true ones obtained with `LinearRegression`. In Figure 6 and Figure 7, we present the same plots obtained with Ridge and Lasso regression, respectively.

In order to compare the methods so far presented we compute for each of them the Mean Squared Error (MSE) and the R^2 values, with and without bootstrapping, and

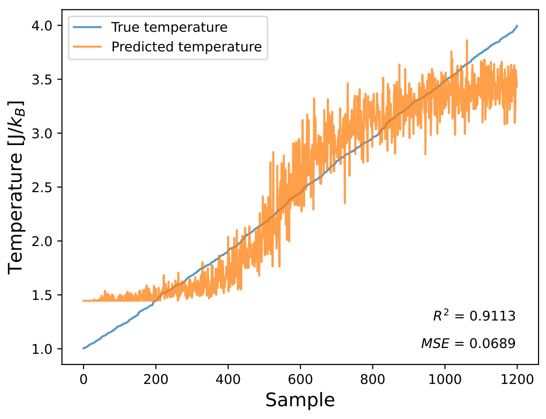

FIG. 4. Prediction over the test set of the temperature given lattice configurations. The predictions were made with Lasso regression and regularization parameter $\lambda = 10^{-2}$. The values of the MSE and R^2 are also indicated.

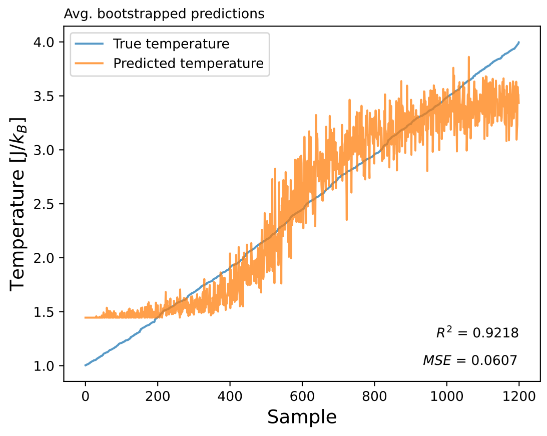

FIG. 6. Prediction over the test set of the temperature given lattice configurations. The predictions have been made with Ridge Regression and averaged over all the 100 bootstraps considered. The values of the MSE and R^2 are also indicated.

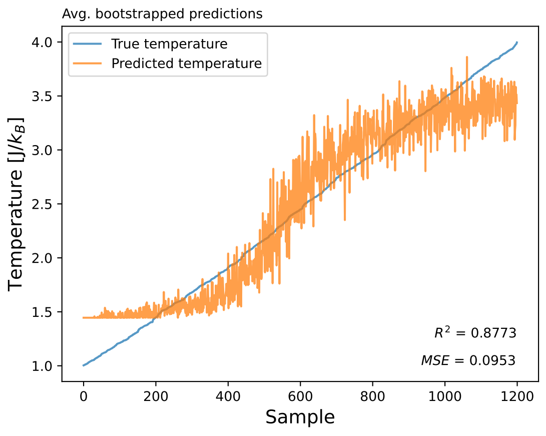

FIG. 5. Prediction over the test set of the temperature given lattice configurations. The predictions have been made with `LinearRegression` and averaged over all the 100 bootstraps considered. The values of the MSE and R^2 are also indicated.

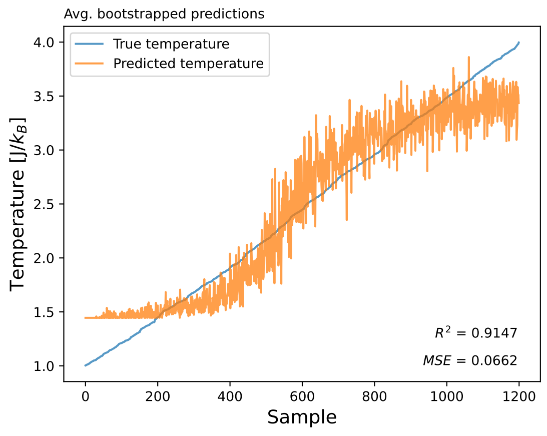

FIG. 7. Prediction over the test set of the temperature given lattice configurations. The predictions have been made with Lasso Regression and averaged over all the 100 bootstraps considered. The values of the MSE and R^2 are also indicated.

collect them in Table III.

	MSE	R^2
OLS	0.0946	0.8782
Ridge	0.0608	0.9217
Lasso	0.0689	0.9113
OLS bs.	0.0953	0.8773
Ridge bs.	0.0607	0.9218
Lasso bs.	0.0662	0.9147

TABLE III. Mean Squared Error (MSE) and R^2 values for the regression methods used, with and without bootstrapping.

Overall we see that Lasso and, particularly, Ridge give the best results. This might be motivated by the fact that these methods introduce a regularization term that penalizes large coefficients in the linear equation. This translates into assigning a different importance to the interactions between the spins.

Furthermore, we notice that implementing the bootstrap does not particularly improve our results. The values of both the MSE and R^2 differ from the "original" one for the third decimal digit. This is expected since our dataset is made of generated data which are by design already balanced.

B. Part II: FFNN

In this Subsection, we show and discuss the result obtained with the use of the implemented FFNN, illustrated in Section II B. In Figure 8, the loss behavior over the training and the validation sets are shown for the number of epochs used. To this regard, we notice how the loss stabilizes already after ~ 10 epochs.

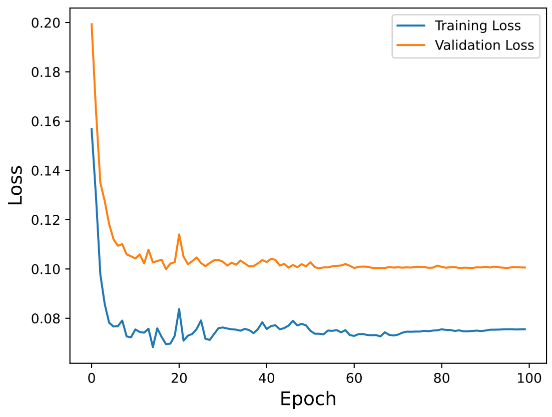

FIG. 8. Loss behavior for training and validations sets over the FFNN implemented, trained over 100 epochs.

In Figure 9, the predicted temperatures together with the true ones and for the test set are shown. In Table IV, we store the computed values of the MSE and the R^2.

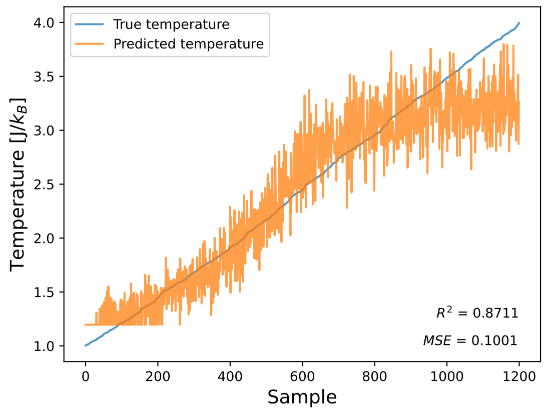

FIG. 9. Prediction over the test set of the temperature given lattice configurations. The predictions were made with the FFNN implemented, trained over 100 epochs.

C. Part III A: CNN for regression

The regression task was done also with the implemented CNN discussed in Section II C. In Figure 10 the loss behavior, for the training and the validation set, over the epochs, is shown.

In Figure 11, we display the plot of the predicted and the true temperature values for the test set.

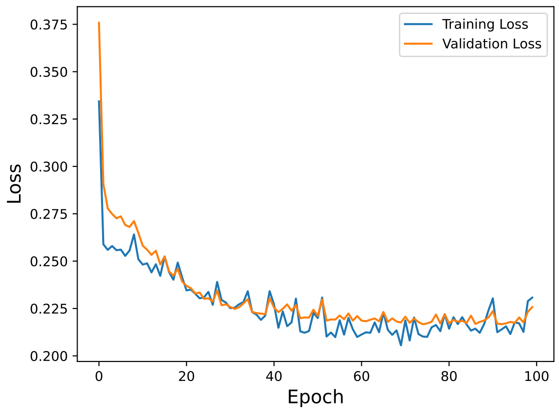

FIG. 10. Loss behavior for training and validations sets over the CNN implemented, trained over 100 epochs.

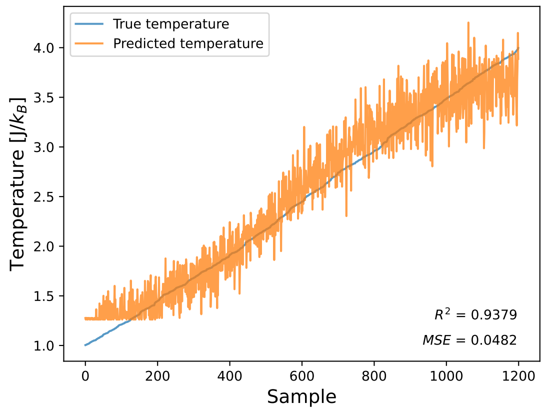

FIG. 11. Prediction over the test set of the temperature given lattice configurations. The predictions were made with the CNN implemented, trained over 100 epochs.

Finally, in Table IV we show the MSE and the R^2 values obtained.

	MSE	R^2
FFNN	0.1001	0.8711
CNN	0.0482	0.9379

TABLE IV. Mean Squared Error (MSE) and R^2 values for the regression task performed with the FFNN and the CNN.

Some overall conclusions can be drawn. By comparing the FFNN and the CNN metrics we witness that the CNN outpeforms the FFNN. The possible reason resides in the kind of data we are feeding to our networks.

In the FFNN's implementation, the matrix X was designed to take into account interactions of each element of the lattice with its neighbors, using their product. This might be quite a big oversimplification, especially considering how a CNN, in comparison, works. Inherently, 2D objects, such as the spin configurations studied in this work, are more suitable to be used with a CNN, which works by design with matrix objects.

Convolutional layers focus on handling the relations between a single element and its neighbors by employing kernels that slide across the input data, capturing local patterns and features. This ability to extract hierarchical representations of features makes Convolutional Neural Networks (CNNs) particularly effective for tasks involving spatial dependencies, and seems well-suited for operating within the nearest-neighbors Ising model framework.

Furthermore, it is interesting to notice that the FFNN performance is strictly connected to its layer size. With the purpose of reaching the best performance we studied the behavior of the metrics over this parameter, and the results are collected in Table V. This highlights how the FFNN performance is generally worse than - or at best similar to (for $N = 2048$) - the regression methods one.

layer size	MSE	R^2
128	0.2675	0.6550
256	0.2395	0.6916
512	0.1720	0.7786
1024	0.1245	0.8268
2048	0.1001	0.8711

TABLE V. Mean Squared Error (MSE) and R^2 values for the regression task performed with the FFNN for different layer sizes.

We notice how by increasing the layer size the FFNN's performance improves, until it becomes comparable to the one generally shown by the regression models. A possible explanation to this phenomenon could reside in the "bottleneck" effect that a small number of neurons might create, considering that our starting number of features is 1600.

D. Part III B: CNN for classification

In this subsection, we analyze the results of the classification task on the phase transition, which consists in classifying the spins' grids as being below or above the critical temperature T_c. This task was carried out with the implemented CNN described in Section II C, with the difference of a sigmoid function as the last layer's activation function.

In Figure 12, we plot the loss behavior for the training and the validation sets. Here, as for the FFNN, ~ 10 epochs seem enough before the training and validation losses stop improving.

In Figure 13, we show the confusion matrix obtained from the model's predictions. From it we can compute

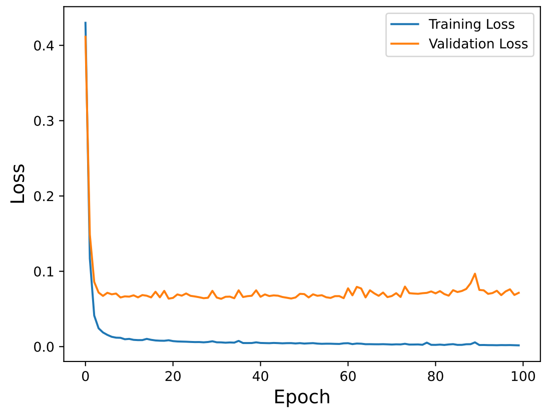

FIG. 12. Loss behavior for training and validations sets over the CNN implemented, trained over 100 epochs. The CNN used has the same architecture as the one in Section III C, with the addition of a sigmoid output layer.

the accuracy of the model as

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}, \tag{22}$$

where TP and TN are the number of true positive and true negative elements respectively, while FP and FN are the amount of false positive and false negative elements respectively. In our case we obtain an accuracy of 98.9% over the test data. Thus, the same CNN architecture used for the regression task revealed to be more than suitable for the classification one as well.

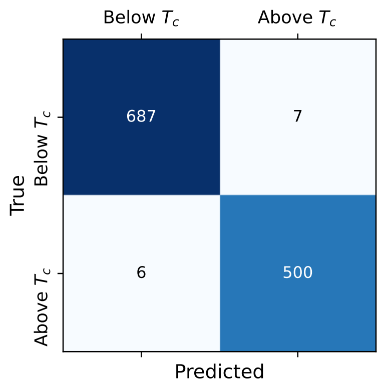

FIG. 13. Confusion matrix from the classification of test lattices as being above or below the critical temperature T_c. The obtained accuracy score over the test set is 0.989.

In Figure 14, we show the plot of the probability of having a temperature value greater than the critical one T_c, for each lattice of the test set. Performing logistic regression on such probabilities allows to find with precision the temperature value for which the model's classification is maximally uncertain, i.e. the value at which $p(T > T_c) = 0.5$, which in our work corresponds to $T_c = 2.273$ J$/k_B$.

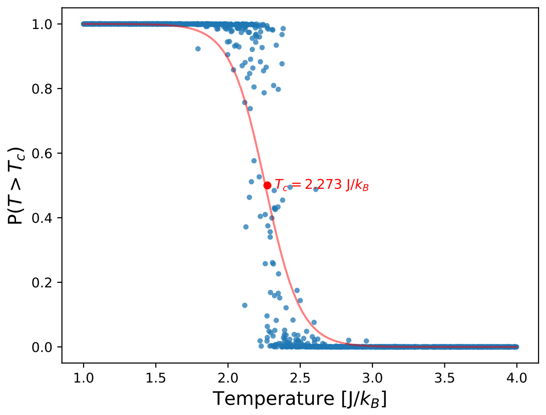

FIG. 14. Classification probability for a given lattice with temperature T of being above the critical temperature T_c. The CNN used has the same architecture as the one in Section III C, with the addition of a sigmoid output layer. Logistic regression (red curve) is performed and, marked in red, is the inferred T_c value.

Despite having worked with relatively small lattices $L = 20$, we found excellent results for the prediction on the critical temperature. The predicted T_c differs from the theoretical value, given in eq.(5), by an error of 0.17%.

IV. CONCLUSION

We have opened this report by introducing the subject of study and the motivations behind this work. Next, we have given a brief overview of the theoretical background of the bidimensional ferromagnetic Ising model. Subsequently, we have described the various ML methods used, starting from regression methods such as OLS, Ridge and Lasso regressions, with some help from sampling methods such as bootstrapping. We then turned to an explanation of FFNNs and CNNs, and for both of the networks proposed we have provided the architecture and discussed the hyperparameters chosen in our implementation. Lastly, we spent some words on explaining how we have generated the dataset used with a Markov Chain Monte Carlo (Metropolis) method.

We have next presented and compared our results, starting from the regression task on the temperature of the spin configuration lattices. At first, we employed the aforementioned regression methods and computed the MSE and the R^2 metrics, with and without considering bootstrapping. Their respective values are collected in Table III from which Ridge regression stands out as the best performing option. Then, we have carried out the same regression task with Neural Networks. We have implemented and trained both a simple FFNN and a CNN and, again, used MSE and R^2 as performance metrics. They are stored in Table IV and V. From those data, we concluded that the CNN, as expected, yields the best results. The quality of the FFNN instead is deeply linked to its layer size, with results comparable with those from the regression models at $N = 2048$ hidden neurons. For all the models used we have visualized the predicted and true temperatures over the test set, highlighting the different performances between the attempted approaches.

Finally, we moved to the classification task of the lattices as being below or above the critical temperature T_c. We have used the same CNN implemented for the regression task with the exception of the last layer's activation function being, here, a sigmoid function. From the confusion matrix, we computed an outstanding accuracy of 98.9%. We have also plotted the probability of a temperature value of being greater than the critical one T_c, for each lattice of the test set. With some help from logistic regression we identified the T_c value as the one for which we have the largest uncertainty in the classification, which resulted in $T_c = 2.273$ J$/k_B$. After a comparison with the theoretical one found by L. Onsager $T_c = 2.269$ J$/k_B$ [15], we concluded that we were able infer the critical temperature with an error of 0.17%.

Altogether we conclude that, when experimenting with 2D Ising Model spins configurations, which can be easily represented as 2D matrix objects, using a CNN guarantees excellent results for both the regression and classification tasks, representing a great choice especially when compared to simpler regression models.

Thanks to the adaptability of ML methods, as well as the Ising model, there are infinite topics and approaches that could be deepened in future works. For starters, an increment in the lattice size L of the generated data, although representing a pretty large obstacle from a computational point of view, could lead to improved and more interesting results. Simulations of the 2D system for a different geometry or even for a dimension greater than two could be also implemented, challenging traditional CNNs. Again, at computational level, with more processing power, the data generation process could be run for even more MC cycles in order to have a better statistics of the system.

Appendix A: Additional plots

In this appendix section we show some additional plots obtained during our simulation.

In Figure 15 and 16, we show the plots of the mean R^2 as a function of the parameter λ for Ridge and Lasso regression, respectively.

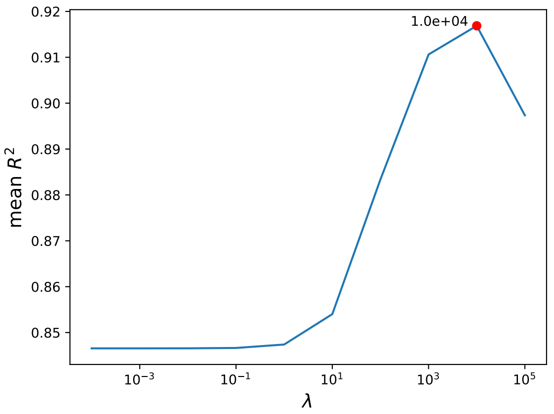

FIG. 15. Mean R^2 as a function of the regularization parameter λ using cross-validation and for Ridge regression. The red dot represents the optimal λ parameter found.

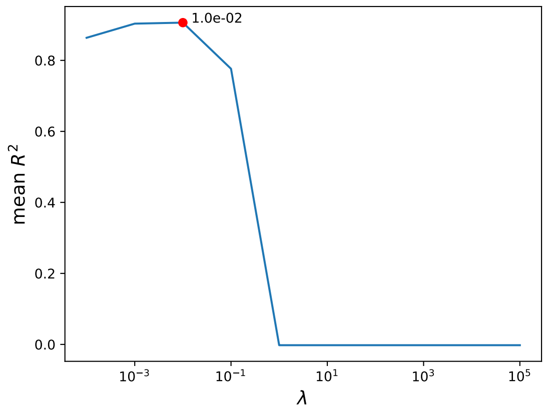

FIG. 16. Mean R^2 as a function of the regularization parameter λ using cross-validation and for Lasso regression. The red dot represents the optimal λ parameter found.

[1] Carleo, G., Cirac, I., Cranmer, K., Daudet, L., Schuld, M., Tishby, N., Vogt-Maranto, L., and Zdeborová, L. (2019). Machine learning and the physical sciences. *Reviews of Modern Physics*, 91(4).

[2] Carrasquilla, J. and Melko, R. G. (2017). Machine learning phases of matter. *Nature Physics*, 13(5):431–434.

[3] Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(61):2121–2159.

[4] Glorot, X., Bordes, A., and Bengio, Y. (2010). Deep sparse rectifier neural networks. volume 15.

[5] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[6] Haas, D. (2022). Identifying ising model phase transitions with neural networks: a motivation for convolutions.

[7] Hjorth-Jensen, M. (2015). *Computational Physics*, chapter 13, pages 415–440.

[8] Hjorth-Jensen, M. (2023). *Notes for the course "Applied Data Analysis and Machine Learning"*. https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/intro.html.

[9] Kingma, D. P. and Ba, J. (2017). Adam: A method for stochastic optimization.

[10] Lecun, Y. (1989). *Generalization and network design strategies*. Elsevier.

[11] Mehta, P., Bukov, M., Wang, C.-H., Day, A. G., Richardson, C., Fisher, C. K., and Schwab, D. J. (2019). A high-bias, low-variance introduction to machine learning for physicists. *Physics Reports*, 810:1–124.

[12] Metropolis, N., Rosenbluth, A. W., Rosenbluth, M. N., Teller, A. H., and Teller, E. (1953). Equation of State Calculations

by Fast Computing Machines. *The Journal of Chemical Physics*, 21(6):1087–1092.

[13] Murphy, K. P. (2022). *Probabilistic Machine Learning: An introduction.* MIT Press.

[14] Newman, M. E. J. and Barkema, G. T. (1999). *Monte Carlo Methods in Statistical Physics*, chapter 3.1.1, page 48. Oxford University Press.

[15] Onsager, L. (1944). Crystal statistics. i. a two-dimensional model with an order-disorder transition. *American Physical Society*, 65(3-4).

[16] Plischke, M. and Bergersen, B. (2006). *Equilibrium Statistical Physics*, chapter 3, page 65. World Scientific.

[17] Setna, J. P. (2017). *Entropy, Order Parameters, and Complexity*, chapter 8, page 163. Clarendon Press.

[18] Shiv Ram Dubey, Satish Kumar Singh, B. B. C. (2022). Activation functions in deep learning: A comprehensive survey and benchmark.

[19] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958.

[20] Tan, H. H. and Lim, K. H. (2019). Vanishing gradient mitigation with deep learning neural network optimization.

[21] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural Networks for Machine Learning,*, 4:26–31.

[22] Zhou, Y.-T. and Chellappa, R. (1988). Computation of optical flow using a neural network. *IEEE 1988 International Conference on Neural Networks*, pages 71–78 vol.2.