Exercise 04 - Supervised learning and Bayesian inference

Federico Agostini, Federico Bottaro, Gianmarco Pompeo

1 Introduction

We are provided with a dataset containing samples generated from the 2D nearest-neighbour coupled Ising model on a 40×40 square lattice; these configurations are at a range of temperatures above and below the critical point.
We wish to classify these states according to their phase using Machine Learning algorithms: with this purpose we will implement a supervised Neural Network and a Random Forest decision tree and compare the results and performances obtained.

2 Brief theoretical overview

The hamiltonian for the Ising model that we are considering can be written in the form

$$\mathcal{H} = -J \sum_{\langle i,j \rangle} S_i S_j \tag{1}$$

with $S_i \in \{\pm 1\}$ where $\langle i, j \rangle$ means we are only choosing values of j that are nearest-neighbour of i; J is an arbitrary energy scale. We will not delve into theoretical details in this context, but it has been proven that this model undergoes a phase transition in the thermodynamic limit from an ordered ferromagnet with all spins aligned to a disordered phase at the critical temperature

$$\frac{T_c}{J} = \frac{2}{\log(1+\sqrt{2})} \approx 2.26 \text{ K} \tag{2}$$

At such threshold the ferromagnetic correlation length diverges; this parameter is what we will have to base our classifications upon.

3 Supervised neural network

The package employed to build our neural network is `TensorFlow`. To begin with, we need to process our data, considering the following three types of states:

- *ordered*, for $T/J < 2.0$;
- *critical*, for $2.0 \leq T/J \leq 2.5$;
- *disordered*, for $T/J > 2.5$.

We use both ordered and disordered states to train the network and, once the supervised training procedure is complete, we will proceed onto evaluating the performance of our algorithm on unseen sub-critical, hyper-critical and critical states. We only use critical states in the test phase because we expect their classification to be computationally more challenging and also to evaluate the predictive power of the algorithm.
We define the loss function to be the cross entropy as predefined in `TensorFlow`, suitable for a binary

classification problem; we minimize the cost function using the Stochastic Gradient Descent (SGD) optimizer.
We now build our Deep Neural Network (DNN) with one hidden layer; we choose the number of neurons in each layer to be $N_{neurons} = 1, 10, 100, 1000$ and the learning rate for the SGD procedure in the range $\eta = 10^{-6}, 10^{-5}, 10^{-4}, 0.001, 0.01$. We will evaluate all possible combinations to study how the performance of the algorithm varies with respect to the parameters.
We train the model using mini-batches of size 100 over a total of 100 epochs; for each trial we retrieve its final loss and accuracy for all three datasets.

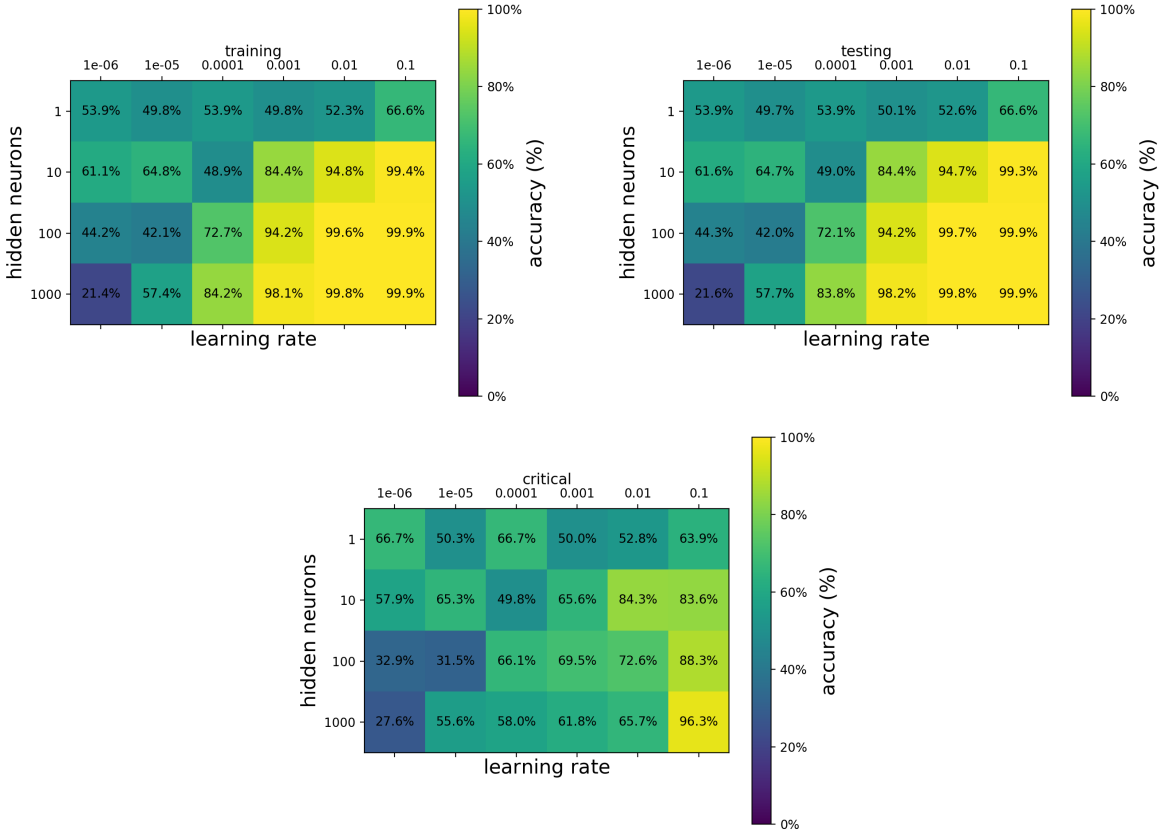

Figure 1: *DNN accuracy score as function of learning rate and hidden neurons.* [Top-Left] *Results for the training set. An accuracy close to 100% is reached for values of the learning rate in range* $10^{-3} \div 10^{-1}$ *even for a small number of neurons.* [Top-Right] *Results for the test set. The same considerations done for the previous case are valid.* [Bottom] *Results for the samples around the critical temperature. An accuracy above 95% is achieved only for a learning rate equal to 0.1 and for a NN with 1000 hidden neurons.*

We emphasize that the Neural Network learns only from the configuration above and below the critical temperature, i.e. where the physical behaviour is well defined. Due to this fact, we get an accuracy of 99% in those regions but also near criticality our network works really well reaching an accuracy of 96,3%, as shown in Figure 1.

4 Random Forest decision tree

For comparison purposes, in this Section we try to perform the same analysis using a random forest algorithm. We use the same dataset and the same criteria to split it; once again we use the critical values only for testing the performance of the algorithm itself.
We will use the `scikit-learn` implementation of random forests. The two main parameters of the random forest are the number of estimators in the ensemble and the depth of the trees used. We

investigate estimators in a range from 10 to 100 with a 10-unit step. As far as the second quantity is concerned, we will just establish how many samples need to be in each node of the classification tree: the bigger this number, the more coarse our trees and data partitioning.
As a simplification, we will just consider extremely fine or extremely coarse trees: both degrees of complexities are sufficient to distinguish between the ordered and disordered phase. When dealing with the critical sample, however, more complex architectures are required.
We now present the performance of hte algorithm, shown in Figure 2 over the training, test and the critical set based on fine and coarse trees at different number of estimators.

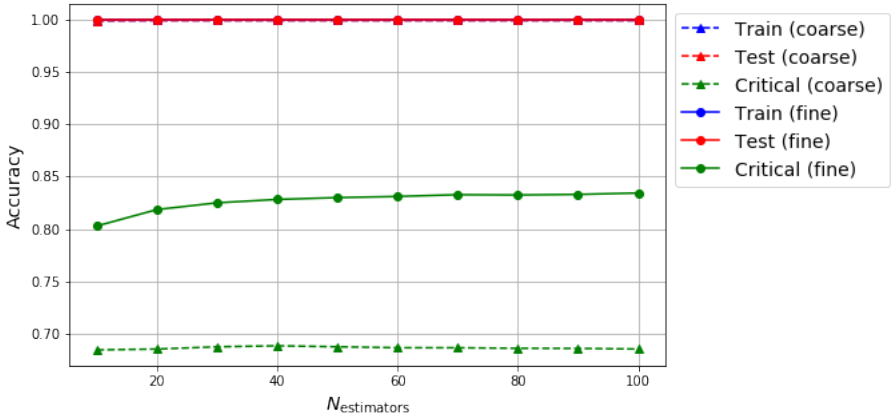

Figure 2: *Performance of the random forest algorithm for the different conditions*[1] *with respect to critical temperature.*

As in the previous section we have that also if the trees are trained below and above the critical temperature, the algorithm is able to reach an accuracy of more or less 85% (considering a fine tree) on this critical region. In training and test we manage to achieve an accuracy of about 100%.

5 Comparison between DNN and RF

We are interested in the comparison between those two learning methods. One of the most meaningful parameter is the accuracy reached: we have that the neural network performs better than the random forest. The former shows a maximum accuracy of 96,3% on the critical set[2], while the latter is below 85%.
Another important aspect is the time taken by the algorithm to perfect this learning. Figure 3 shows the time that both methods need to perform the learning process. Also under this criterium the neural network proves to be better: in fact the network with 1000 hidden neurons and a learning rate of 0.1 (that is the one that reaches the highest accuracy) employs only 6.82 seconds while the random forest with 100 estimators around 22 seconds, which is about 3 times as long.

6 Bayesian approach

We tackle the same classification problem this time using a Bayesian approach, at least from a theoretical standpoint. This implies a radical change of perspective, in that the unknown set of parameters are treated as random variables instead of as a set of fixed yet unknown values.
Bayes' theorem states that

[1] Please note the blue lines are almost entirely overwritten by the red ones.
[2] We use the results obtained at criticality as a mean of comparison given that neither algorithms used it in training and therefore it measures, in a way, the learning ability of the methodology employed.

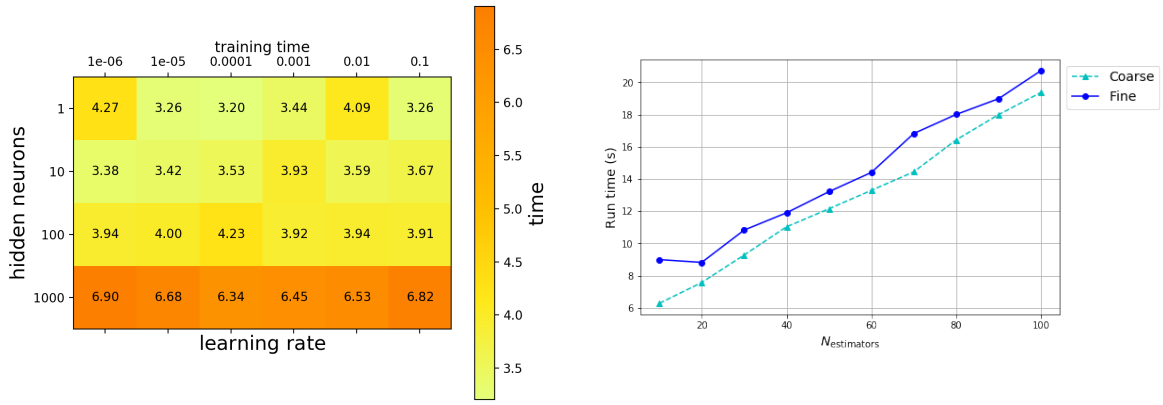

Figure 3: *Run-time for the training of the algorithms used for the classification.* [Left] *Run-time for the DNN as function of the learning rate and the number of hidden neurons; the result is heavily influenced by the number of neurons in the network.* [Right] *Run-time for the RF as function of the number of estimators. Two lines for a coarse and a fine tree are plotted to show the difference in execution time.*

$$P(M|D) = \frac{P(D|M)\mathcal{P}(M)}{P(D)} \tag{3}$$

where the conditional probability $P(M|D)$ is known as posterior, $P(D|M)$ is the likelihood, $P(M)$ the prior and $P(D)$ the evidence.

The likelihood can be chosen as the product of the energies of the different configurations, normalized by the partition function

$$P(D|M) = \frac{\prod e^{-\beta \mathcal{H}_i}}{\mathcal{Z}} = f(T, J, \{\sigma_i\}) = f(\tilde{J}, \{\sigma_i\}) \quad \tilde{J} = T/J \tag{4}$$

with evident choice of notation.

The prior is the quantity that encapsulates our knowledge on the system; to do so, we have a certain degree of flexibility, since there is no right or wrong choice, just more suitable ones. What we do know is that the system can only achieve the states 0 or 1, so we could pick a uniform distribution between these two extremes. This is usually the most natural choice, but in this instance it seems like an excessively naïve one since we do know our variable is discrete in the chosen interval. We therefore opt for a combination of two gaussians centered in 0 and 1 respectively.

Using two normal distributions we also achieve the advantage of having a continuous variable which in a way recalls the magnetization, instead of just working with a binary classification.

The evidence, although difficult to compute, has just the role of a normalization factor and thus it can be represented as a constant value, say Ω.

Another way to proceed could be using two different features, which can be extracted from the configurations: the magnetization M and the two points correlation function G, that can be seen as a sort of "error" in the value of M. In the ordered phase, we expect $|M| \sim 1$, with a small value for G, while near the critical point we get $|M| \sim 0$ and a large value of the correlation function (that should diverge at $T = T_c$ exactly). Using this two features, we can train an algorithm such as a Support Vector Machine, and we can implement a Bayesian approach by adding a regularization term (of the kind L_1 or L_2), in order to perform a Maximum a posteriori probability estimate.