group2312 - Exercise 2 – Performance analysis of a Restricted Boltzman Machine

Alessandro Breccia, Marco Chiloiro, Riccardo Lui, and Konstantinos Panagiotakis
(Dated: April 2, 2023)

In the paper, we present the results of a Restricted Boltzmann Machine (RBM) applied on a binary dataset. This machine is an artificial neural network capable of generating new data according to the given input. The probability distribution of the data is not known to the algorithm. Thanks to auxiliary 'hidden' units, the model allows to recognize the correlations between the variables and recreating data according to the unknown pdf. We analyzed the performance of our machine while changing the parameters of the training procedure. Different methods are proposed to verify the quality of the training and generated dataset.

INTRODUCTION

A *Restricted Boltzmann Machine* (RBM) is an energy-based generative neural network, consisting of an input layer, called *visible layer*, and an *hidden layer*. Through the interaction between visible variables v and hidden variables h, the model is able to simulate the probability distribution underlying the input dataset and extrapolate high-order correlations.
This process begins with mapping the problem in a physics environment: the Ising model. It consists in a lattice where each site is a variable, called spin, having 2 possible values. The spins tend to be aligned or anti-aligned depending on the coupling constant J_{ij} between v_i and v_j. The effects of a possible field $\vec{f}$ that influence the spins and the tendency to align can be described by the energy of the system:

$$E(\mathbf{v}) = -\sum_{i,j=1}^{N} J_{i,j} v_i v_j - \sum_{i=1}^{N} f_i v_i. \tag{1}$$

Writing $J_{i,j}$ as $\sum_{\mu}^{N} W_{i,\mu} W_{j,\mu}$, we can decouple the dependency between v_i and v_j, performing an Hubbard-Stratonovich transformation, introducing auxiliary variables called h [1]. The reason for this transformation relies on the fact that now the interactions are only between the two different layers and this can be seen in the new shape of the energy:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i,j=1}^{N} v_i W_{i,j} h_j - \sum_{i=1}^{N} a_i v_i - \sum_{j=1}^{N} b_j h_j. \tag{2}$$

Notice the choice of Bernoulli layers, i.e. binary vectors and linear biases a_i and b_j. W is an $L \times M$ matrix where $W_{i,j}$ gives the weight of the interaction between v_i and h_j. L is the number of visible units and M is the number of hidden units. The biases $\vec{a}$, $\vec{b}$ are respectively vectors of length L and M.

METHODS

The dataset that we used was comprised of N=10^5 samples, where each sample consisted in a line of $L = 20$ units that were generated following the one hot encoding rule, which consists in producing 5 blocks of 4 units, in which only one unit is activated, i.e. set to 1 . A block can be subdivided into two sub-blocks of 2 units, a right one and a left one. A block with an activated unit in its sub-block on the left represents a "positive" block, **P**, whether on the right represents a "negative" one, **N**. The dataset we are working on follows this alternating pattern of $\mathbf{P}-\mathbf{N}-\mathbf{P}-\mathbf{N}-\mathbf{P}-\mathbf{N}$ of sub-blocks with a 10% error, introduced to represent random noise.

Our RBM is composed by a visible layer of size $L = 20$ units, and a hidden layer of size $M = 6$. This structure of the RBM, with $M < L$, forces the model to be so simple that only the crucial information and patterns are captured, avoiding overfitting as much as possible. The first step starts from the visible layer v, which encodes the correlations between its units in h, activating each hidden variable with probability:

$$p\left(h_j = 1 \mid \mathbf{v}\right) = \frac{1}{1 + e^{-GAP\left(b_j + \sum_{i=1}^{L} v_i W_{ij}\right)}}, \tag{3}$$

using a Markov-Chain-Monte-Carlo process (MCMC) through a Sigmoid function as activation. MCMC is also used to compute the averages of the generated data and more details about the procedure can be found in [2].
Once the hidden layer is activated, the model generates a new data sample, starting from a line of zeros of length L and turning to 1 each unit of the fantasy variable v_i^m with probability:

$$p\left(v_i = 1 \mid \mathbf{h}\right) = \frac{1}{1 + e^{-GAP\left(a_i + \sum_{j=1}^{M} W_{ij} h_j\right)}}. \tag{4}$$

This procedure can be iterated several times, creating from the fantasy layer v^m and a hidden layer h^m: these last two can be seen as the model layers, while the first v and h are the data layers. This technique is called Contrastive Divergence (CD), and can be composed of n steps (which is typically close to 1), leading to a $CD-n$.

To obtain the best parameters we implemented a Maximum Log-Likelihood Estimation (MLE) [1] to define a cost function ($C = -$Log-Likelihood) which have to be minimized. In our case, we directly maximize the Log-Likelihood.

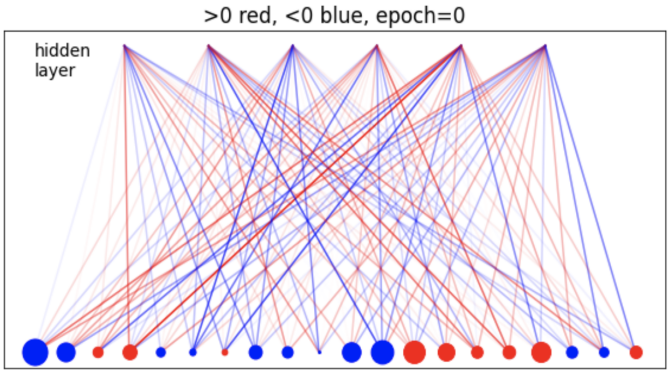

FIG. 1. Graphical sketch of RBM with $M = 6$ and $L = 20$. Red refers to a positive interaction between data and weights for the links and between data and biases for the units, while blue refers to negative interactions. The thickness of the links and size of the units represents the strength of the interaction.

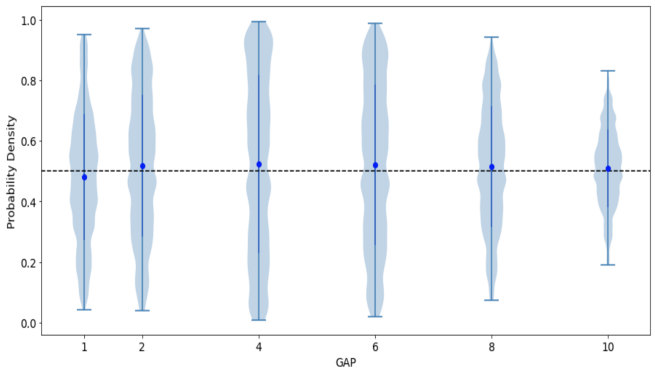

FIG. 2. Violin plot of probability density distributions given by the activation function, i.e. the sigmoid function, applied to the initial weights and biases with GAPS = [1,2,4,6,8,10], using bits [0,1],[-1,1],[-2,2],[-3,3],[-4,4],[-5,5]. The distributions range between uniform to gaussian, correcting the bimodal distribution seen in the original [-1,1] set up.

Calling the parameters θ_i and defining the Log-Likelihood:

$$\mathcal{L}(\theta) =< \ln(p_\theta(x)) >_{\text{data}} = - < E(x)_\theta >_{\text{data}} - \ln(Z_\theta), \quad (5)$$

and the operator $O_i(\mathbf{x}) = \frac{\partial E_\theta(\mathbf{x})}{\partial \theta_i}$ one can show that

$$-\frac{\partial \mathcal{L}_\theta(x)}{\partial \theta_i} = \langle O_i(\mathbf{x}) \rangle_{data} - \langle O_i(\mathbf{x}) \rangle_{model}. \quad (6)$$

In our case $\theta_i = \{\mathbf{a}, \mathbf{b}, W\}$, so that deriving (2) with respect to the three parameters one gets:

$$\frac{\partial \mathcal{L}(W_{i,\mu}, a_i, b_\mu)}{\partial W_{i,\mu}} = \langle v_i h_\mu \rangle_{data} - \langle v_i h_\mu \rangle_{model} \quad (7)$$

$$\frac{\partial \mathcal{L}(W_{i,\mu}, a_i, b_\mu)}{\partial a_i} = \langle v_i \rangle_{data} - \langle v_i \rangle_{model} \quad (8)$$

$$\frac{\partial \mathcal{L}(W_{i,\mu}, a_i, b_\mu)}{\partial b_\mu} = \langle h_\mu \rangle_{data} - \langle h_\mu \rangle_{model}. \quad (9)$$

The starting point to apply the gradient ascent algorithm is to initialize the parameters $\{\mathbf{a}, \mathbf{b}, W\}$: while in the second one every component is set to 0, the first and the last one are initialized following a Gaussian distribution with mean $\mu = 0$ and variance $\sigma = 1$, multiplied by a constant $\kappa = \frac{2}{\sqrt{(M+L)g(M,GAP)}}$ [1]. The $GAP = v_{max} - v_{min}$ and the function $g(M, GAP)$ are used to map the variance to system defined on spins values $\{-\alpha, \alpha\}_{\alpha=1,2,..}$. Since the dependency on both M and GAP is complex to extrapolate without strong basis on κ, we focused on $M = 6$ and found out that $g = e^{GAP}$ could represent a good correction for $\alpha \in [1, 5]$. From now on we applied our study on the binary case { 0,1}. In fig.1 we show a visual representation of our setup.

The gradient ascending procedure that updates the parameters was implemented with 4 different kind of optimizers: the standard Stochastic Gradient Ascend (here called "Vanilla"), Adam, RMSprop and the so Centering Trick[3]. Every optimizer works on mini-batches of 500 samples. After each epoch, the dataset is shuffled to randomize the process.
The main differences between the 4 different updating techniques rely on the second moment of the gradient: while Vanilla and the Centering Trick take into consideration only the first moment of the gradient, Adam and RMSprop sum up also the contribution of the second moment, together with a rescaling of the update vector, to avoid the explosion or vanishing of the gradient.
The Centering Trick is a special optimizer used in RBM to add an offset $\vec{\mu} = (\mu_1, ..., \mu_L)$ and $\vec{\lambda} = (\lambda_1, ..., \lambda_M)$ respectively to the visible and hidden unit, with the aim of making the RBM invariant to flip transformations if a flip of v_j to $1 - v_j$ implies a change of μ_j to $1 - \mu_j$ and a flip of h_i to $1 - h_i$ implies a change of λ_i to $1 - \lambda_i$ [3].

Once the analysis with the standard activation function was conducted, in order to increase the efficiency of the algorithm, we forced the backward Contrastive Divergence Step by imposing the One-Hot Encoding pattern. We implemented a new activation function, for the backward Contrastive Divergence Step, that computes the contribution of the weights and biases by groups of 4, picking from the 4 possible choices of blocks the one with minimum energy with higher probability. This process, from now on called Constrained Backward Step (CBS), consists in calculating the local field applied to the i-th group of 4 composing each sample, with $i = \{1, 2, 3, 4, 5\}$, generated by the hidden layer. To define the energy associated to the j-th block b^j, with $j = \{1, 2, 3, 4\}$, we perform the product between the first group of v, v^i, and the W matrix reduced from dimension $L \times M$ to

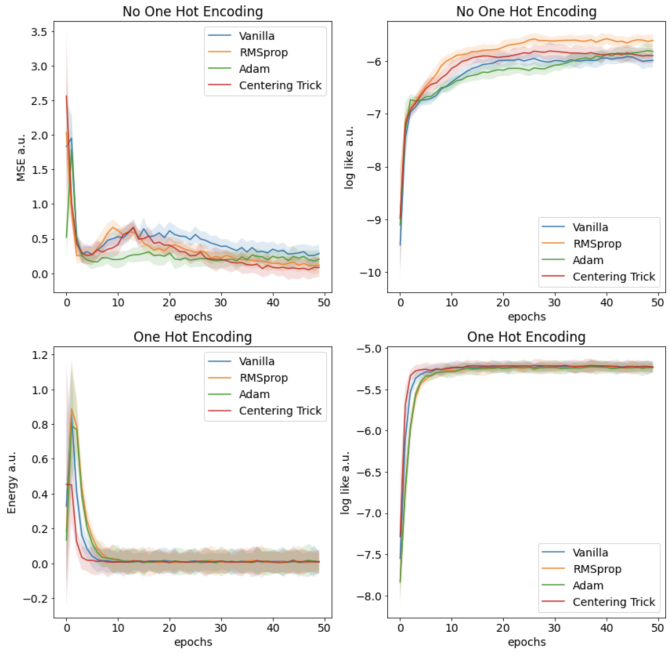

FIG. 3. MSE (first column) and Log-Likelihood (second column) for Vanilla, RMSprop, Adam and Centering Trick optimizers with and without CBS, in the figure named as "One-Hot Encoding". The data are reported as the mean value over the mini-batches for each epoch with the corresponding std shaded around the mean value.

$4 \times M$, called here W_i^{red}, where the indexes of the 4 rows correspond to the ones of the i-th group. The energy associated to

the j-th block is given by:

$$E_j^i = -\sum_i^{i+3}[\sum_\mu^M W_{i,\mu}^{red} h_\mu + a_i] \cdot b^j. \tag{10}$$

After having computed E_j^i, each probability to place the respective block is given by the Boltzmann distribution $p_j^i = \frac{e^{-E_j^i}}{Z_i}$, where $Z_i = \sum_{j=1}^4 e^{-E_j^i}$ is the partition function. As in standard MC processes[2], we then draw a random number x between 0 and 1 and select the block whose range in the cumulative probability include x.

To evaluate the performances of our RBM setups, we used a set of quality indicators:

- Log-Likelihood as a function of the epochs (LL)
- Mean Squared Error (MSE)
- Second Moment Error (SME)
- Adversarial Accuracy Indicator (AAI)
- Percentage Error (PE)

The LL indicator plays the role of the opposite of a loss function in a standard machine learning problem, so that its behaviour gives a feedback about the goodness of the training procedure.

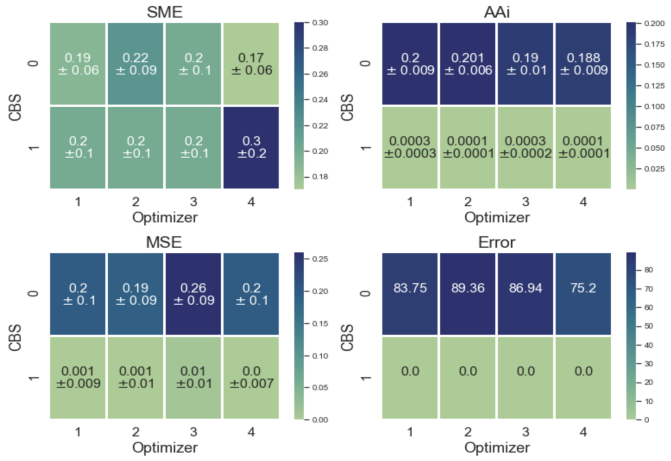

FIG. 4. SME, Mean Squared Error (MSE), Percent Error (Error), AAI as a function of contrastive divergence steps in the range from 1 to 4 for the RSMprop optimizer with and without CBS. For the cases with CBS we consider $M = 2$, while for the cases without CBS, $M = 6$.

The mean squared error, or MSE, is a measure of the average of the squares of the differences between the energy of data and model.

In a well trained algorithm, the MSE should tend to 0 as the training procedure develops.

The Second Moment Error, or SME, computes the averaged covariance matrix of the data representation sites C_{ij}, for both data and model. Once having both matrices we estimate the distance $\mathcal{E}_c$ between both kind of C_{ij}:

$$\mathcal{E}_c \equiv \frac{2}{N_{\rm v}\,(N_{\rm v}-1)} \sum_{i<j} \left(C_{ij}^{\rm RBM} - C_{ij}^{\mathcal{D}}\right)^2 \tag{11}$$

where N_v is the size of sample used.

The Adversarial Accuracy Indicator, or AAI, evaluates how much the the generated data samples are similar to the original ones. Here the distance introduced is simply the sum of the absolute value of the difference between each component of the sample. In an optimal model, A_s and A_t should tend to $\frac{1}{2}$, so that $\mathcal{E}_c = (\frac{1}{2} - A_s)^2 + (\frac{1}{2} - A_t)^2$ tends to 0. More details about these indicators are explained in reference [4].

The percent error, or PE, checks the preservation of the alternation of the one-hot encoding pattern ,*P-N-P-N*, in the sample recreated at low temperature, i.e. with a large amplitude in the activation function in order to force the emergence of the main pattern learned by the model.

RESULTS

The main aim is to achieve a well trained RBM in order to being able to visualize the one-hot encoding pattern

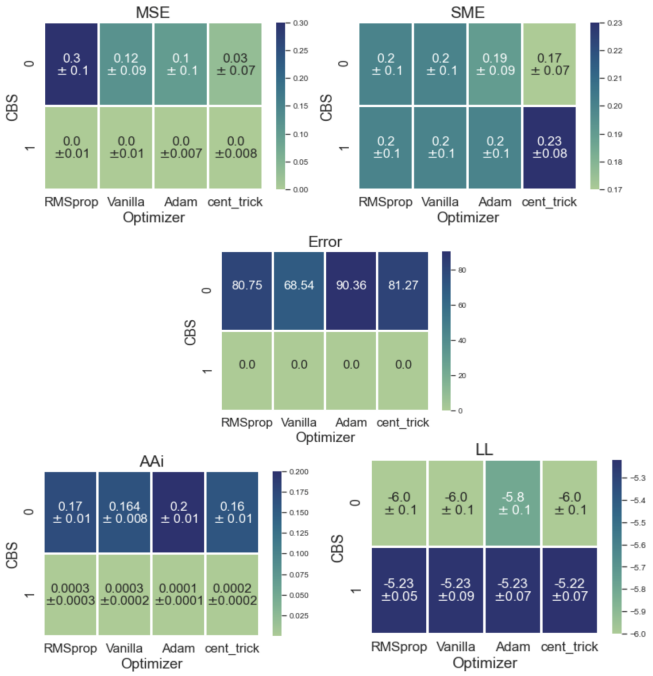

FIG. 5. SME, AAI, Mean Squared Error (MSE), Percentage Error (PE), Log-Likelihood for Vanilla, RMSprop, Adam and centering trick optimizers with and without CBS. In the LL, we focused our analysis mostly on its general behaviour rather than the actual final value, which differs from $M = 6$ to $M = 2$.

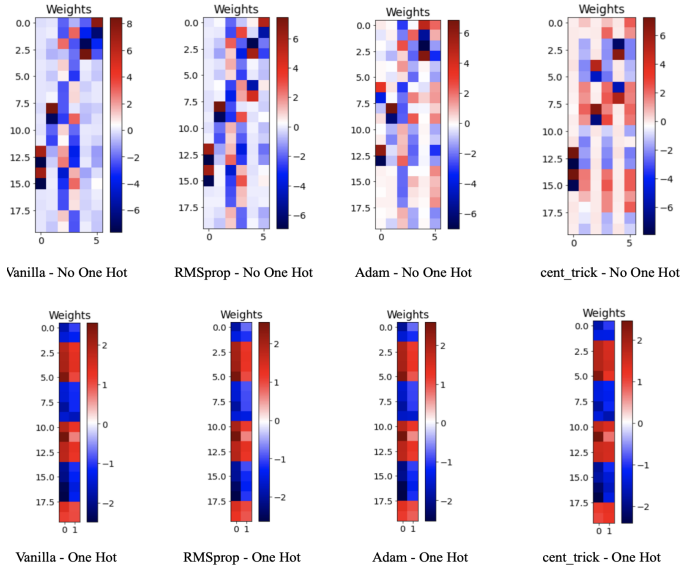

FIG. 6. Weights strength (lighter colors refer to weaker weights, whether darker colors refer to stronger weights) and sign, (red corresponds to a positive weight, whether blue corresponds to a negative weight). The combination of weights is for Vanilla, RMSprop, Adam and centering trick optimizers with ("One Hot") and without CBS ("No One Hot"). The number of columns M refers to the depth of the hidden layer, and the number of rows L refers to the depth of the visible layer.

underneath the input data structure. Switching between optimizers and activation functions, and validating the process through the indicators, we managed to check the quality of the training. Let's now dive in deeper and describe each one of the main results that were obtained for each executed procedure.

Figure 3 shows the LL and MSE for all four used optimizers during the training of the RBM, both with and without CBS. In the case with CBS, the MSE starts at a lower value, decreasing to zero in fewer epochs than in the case without CBS, hence allowing for the Log-Likelihood to be maximized also in fewer epochs, as we can see from equation (5). Given this relationship, we can conclude that the centering trick was the best performing optimizer out of the four tested ones given our initial parametrization of the setup. Furthermore, the training sessions without CBS show a more unstable and oscillating behaviour after few epochs, suggesting that the learning task is not able anymore to perform efficiently, due to the complexity of the problem and to the activation function used. Figure 4 displays indicators as a function of increasing contrastive divergence steps. In most of the cases, all the shown indicators give better results with CBS. Looking at the indicators' heatmap, we can conclude that the number of CD_{steps} is quite irrelevant in the CBS case, so the best choice would be $n = 1$ because it is computationally more efficient with respect to the other cases, as it has to recreate the dataset only once for each sample in the cycle. Without CBS, increasing the steps, the performances seems to get slightly better after 3 steps, but the average values are compatible with the ones with $n = 1$. Figure 5 shows the performance indicators as a function of the optimizers used. For the SME, Adam, RMSprop and Vanilla are aligned on the same value of 0.2 ± 0.1 for the case with CBS, whether the Centering Trick has the lowest SME with a value of 0.17 ± 0.07 for the case without CBS. For the AAI indicator, with CBS, Adam has the best output with a value of 0.0001 ± 0.0001, whether without CBS, the centering trick is the best one with a value of 0.16 ± 0.01. As the Error, or PE shows, we can re-iterate that the performance of the RBM with the CBS, has the best results. The lowest error without CBS is given by the Adam optimizer. Looking at the MSE, the centering trick has the best results without CBS with a value of 0.03 ± 0.07. For the Log-Likelihood with CBS, the centering trick has the highest one, whether without CBS, Adam has the best result. In conclusion the pattern that emerged suggests that the centering trick is the most suitable optimizer for a RBM with a standard activation function, while, imposing the CBS, the best optimizer is Adam, with slightly lower indicator values, with respect to the other optimizers, and less instability.

Figure 6 shows the weigths values W_{ij} for all the optimizers with CBS (here called "One-Hot"), with a dimension of 20×6, and without CBS, with a dimension of 20×2. The colors in the plot highlight the pattern learned by the machine, with alternating positive and negative weights that suggest the one hot encoding structure represented in the dateset. In the case without CBS, with the centering trick, it is clear how most of the hidden units impose a clear common pattern, while the other optimizers have found different configurations from the centering trick one, but all quite similar one to another. In all these configurations mainly 2 hidden units act on the total length of the sample, while the other act locally. This could represent a key point in the implementation of a better architecture and training procedure for an optimal RBM with a standard activation function. With CBS all the optimizers highlight a common pattern related to the one hot encoding structure.

Overall, the training of the RBM using the CBS constraint had by far the best performances in each one of the provided indicators, implying that the activation function plays a key role in the Neural Network, enabling it to learn the pattern in the most efficient way.

CONCLUSIONS

We introduced and showcased the performances of a Restricted Boltzmann Machine, applied on a binary dataset that allowed us to generate a modelled dataset from the original one. The neural network worked efficiently with the implemented CBS, allowing the machine to capture accurately the probability density underlying the input dataset. This can be seen by imposing the machine to reconstruct the dataset without any noise, allowing for the emergence of the main structure of the dataset, producing the one-hot encoding pattern without any errors. Without the constraints, the RBM failed to reconstruct the main pattern but gave hints about the possible structure of the input samples, especially using the Centering Trick, as we can see from fig.6. To sum up, the RBM is a straightforward unsupervised machine learning method that retrieves pre-existing results and improves them. Moreover the RBM design can be interpreted and read more easily with respect to other Generative Neural Networks. [5]

[1] P. Mehta et al, A high-bias, low-variance introduction to Machine Learning for physicists, 86–94 (2019).

[2] MacKay, David JC Information theory, inference and learning algorithms, (2003) ch. 29-31

[3] Bortoletto, Matteo Study of performances for Restricted Boltzmann Machines, (2020/2021) pg. 28-29

[4] Aurélien Decelle et al, J. Stat. Mech. (2022) 114009

[5] Anna Braghetto, Enzo Orlandini, Marco Baiesi (2023), "Interpretable machine learning of amino acid patterns in proteins: a statistical ensemble approach", arXiv:2303.15228v1 [q-bio.BM]