Group 2301 – Restricted Boltzmann Machines

Antonio Feltrin, Giosuè Sardo Infirri, Riccardo Tancredi, and Simone Toso
(Dated: April 2, 2023)

In many physics problems the collected data is affected by a certain amount of noise. If this noise is strong, we may want to reduce it before data analysis. Restricted Boltzmann Machines (RBMs) are generative models which prove to be useful for denoising, while at the same time providing useful insight into the correlations behind the data. In this work, we train an RBM through different implementation setups, with the aim of analysing the structure of some known and unknown data sets. Then we visualise the weights of the RBM to see if it effectively understands the pattern behind the data and, finally, we use the trained model to generate virtual data which are not affected by the random noise of the original data set.

INTRODUCTION

The aim of the project is to train an RBM on two different data sets, whose encoding is inspired by the minimal structure of a protein. We have two data sets available, one of known and the other of unknown structure. In both data sets the information is represented through one-hot encoding.

In the first data set, each datum consists of $G = 5$ blocks of $A = 4$ bits. A block can be either P (1000 or 0100) or N (0010 or 0001). The data are generated by choosing either a $PNPNP$ or $NPNPN$ pattern. With probability $q = 0.1$ a nibble can randomly change state; if this happens, the nibble assumes one of the four possible one-hot encodings with uniform probability. This is the source of noise in the data. We expect the RBM to detect the encoded pattern, i.e. the 1 alternating from left to right. Furthermore, by generating new data through the RBM and setting a low temperature - i.e. high amplitude in the exponent of the Boltzmann factors - we can perform a denoising on the original data set, reducing the random fluctuations.

For the second set, we do not know what pattern was followed when generating the data: by training the RBM and visualising its weights, we will be able to decode it.

METHODS: PERFORMANCE EVALUATION

In the following pages, we will refer with v_i to the i-th visible unit and with h_j to the j-th hidden unit. We will then define the weight of the interaction between v_i and h_j as $w_{i,j}$, the bias of v_i as a_i and the bias of h_j as b_j.

In order to judge the performances of the Boltzmann Machine, we have introduced some specific evaluating functions.

Our-score Since the generation process of the first data set is given, we know what pattern lies behind it. Therefore, to evaluate the data denoising performed by the RBM, we compute the fraction of the generated data which presents the right encoding. Only perfectly alternating ($PNPNP$ or $NPNPN$) lines of data are considered valid and this function returns the fraction of correct lines. `Our_score` is based on a supervised learning approach, so we do not rely completely on it.

Log-likelihood An unsupervised approach is provided instead by the log-likelihood $\log \mathcal{L}$. Given the energy

$$E = -\sum_{i,j} v_i w_{ij} h_j - \sum_i a_i v_i - \sum_j b_j h_j \tag{1}$$

we define the log likelihood as:

$$\log \mathcal{L} = -\langle E(\mathbf{v}, \mathbf{h}) \rangle - \log(\mathbf{Z}) \tag{2}$$

where $\langle E(\mathbf{v}, \mathbf{h}) \rangle$ is the arithmetic average of the energy across the data set. The partition function is defined as $Z = \sum_{v,h} e^{-E(v,h|\theta)}$ across all possible combinations of v and h, with $\theta = \{\mathbf{w}, \mathbf{a}, \mathbf{b}\}$. During training, the RBM tries to maximize this function by following its gradient. The evaluation of the partition function is computationally feasible with the one-hot encoding representation, since we have $A^G \cdot 2^M = 4096$ total configurations (with $M = 2$).

Adversarial Accuracy Indicator (AAI) This third function is an alternative error function: it allows to see if the generated data could be overlapped to the real set [1]. In order to measure this quantity, one has to introduce two parameters, A_S and A_T, that evaluate the average type of the nearest point:

$$A_S = \frac{1}{N_S} \sum_m^{N_S} \mathbb{1}\left(d_{SS}(m) < d_{ST}(m)\right) \tag{3}$$

$$A_T = \frac{1}{N_S} \sum_m^{N_S} \mathbb{1}\left(d_{TT}(m) < d_{TS}(m)\right) \tag{4}$$

In fact one can expect those parameters to tend to $\frac{1}{2}$ for a model that perfectly emulates the data. Finally, we compute the error as $\mathcal{E}_{AAI} = (A_S - 1/2)^2 + (A_T - 1/2)^2$.

Entropy The last adopted indicator is the entropy, which tends to 0 if the original and model-generated data are similar. The idea is to see the length of the compressed data using the function `gzip`. The entropy of N_S samples of the data set is S_{src} and the entropy of a set composed merging half of the the data set samples and half of the one coming from the generated data is S_{cross}.

The entropy difference is therefore defined as in [2]:

$$\Delta S = \frac{S_{cross}}{S_{src}} - 1 \tag{5}$$

In general, one expects a high entropy for a generated data set that is more disordered than the original one and a small entropy otherwise.

RBM TRAINING

In this core section we search for the best architecture for our RBM among a combination of different hyperparameters:

- **Data representation (SPINS):** this can be either $[0, 1]$ or $[-1, 1]$, thus mapping each 0 of the original data set to a -1.
- **CD-steps**: the number of Contrastive Divergence steps. We will choose among $\{1, 5, 10, 20\}$.
- **GD algorithm**: how we apply gradient descent, which is either through **vanilla SGD** with an adaptive learning rate or by implementing the **Adam** algorithm.

We search for the best set of hyperparameters by training the RBM on the **first data set**. We force the RBM to generate the data using one-hot encoding, thus reducing the number of visible configurations from 2^{AG} to A^G. The size of the minibatch is kept fixed at 16.

Simulations are run for 100 epochs since, as we will see, this is usually enough for the training to stabilize. For all hyperparameters we kept a number of 2 hidden units. To be precise, given the nature of the pattern, one unit should have been enough. However, we preferred keeping more units to allow for more variability in the behaviour of the RBM.

In Figure 1 we compare the AAI scores after 100 epochs for the various hyperparameters.

As we can see, there are multiple valid choices for the hyperparameters. The AAI is lower when using $[0, 1]$ representation, compared to the $[-1, 1]$. The best value is obtained when implementing **Adam** with data in the form $[0, 1]$ and CD-20. From the theory we would expect a similar behaviour, as the increasing number of Contrastive Divergence steps should better represent the data.

In order to show the good behaviour of the optimal configuration, we study the log-likelihood as a function

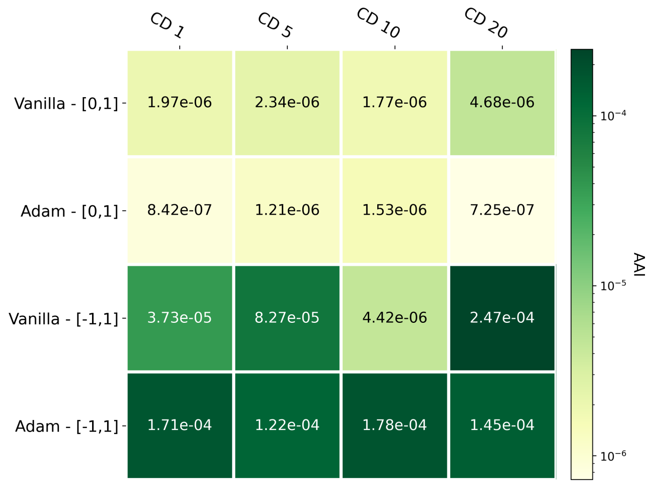


Figure 1. Heatmap of the AAI for each of the considered hyperparameters (logarithmic scale on the colormap).

of the epochs, along with the A_S and A_T scores which define the AAI index. To get more statistically varied samples, we run the program using five different random seeds. The result is shown in Figure 2.

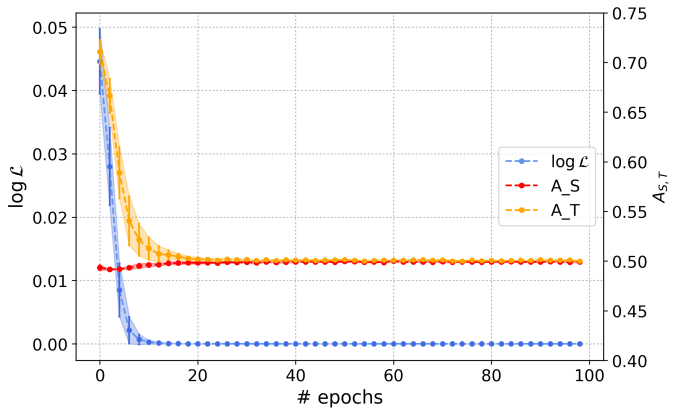


Figure 2. Plot of $\log \mathcal{L}$, A_S and A_T as a function of the epochs, using the best configuration found. The shaded colors indicate the standard deviation accross the performed simulations.

As shown in Figure 2, the quantities grow rapidly and reach a plateau around the 20-th epoch. The A_S and A_T indicators stabilize around the theoretically expected value of 1/2.

For further considerations, we show the behaviour of the other two estimators presented in the previous section: `Our-score` and the entropy. For the evaluation of `Our_score`, we kept a high temperature (amplitude of 1 in the Boltzmann weights).

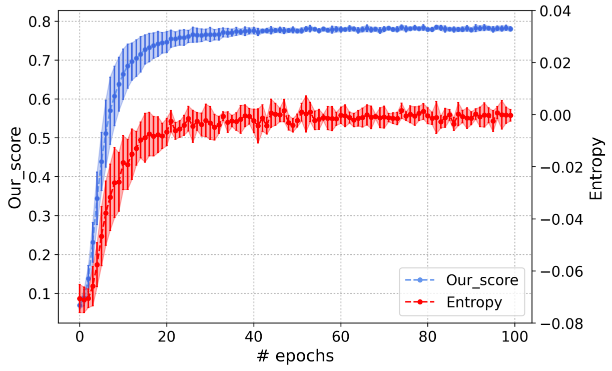


Figure 3. Plot of the score and of the entropy as a function of the epochs.

The plots in Figure 3 confirm the convergence of the algorithm after approximately 20 epochs. Furthermore, we see that, during training, `Our-score` stabilizes at the value of 77%. This is an interesting result that is worth reflecting upon.

We know that we generated the data by imposing alternating polarity, but allowing for random errors for each nibble with probability $q = 0.1$. When the nibble randomly assumes a new encoding, there is still a 50% probability that it maintains the correct polarity, since the 1 can fall either on the left or on the right side. Therefore, the probability for a single datum in the original data set to be correct is

$$\begin{aligned} p(q) &= \left[\sum_{k=0}^{5} \binom{5}{k} q^k (1-q)^{5-k} \left(\frac{1}{2}\right)^k\right] + \left(\frac{q}{2}\right)^5 \\ &= \left(1 - \frac{q}{2}\right)^5 + \left(\frac{q}{2}\right)^5 \end{aligned} \tag{6}$$

The term $(\frac{q}{2})^5$ comes from the fact that, when all the nibbles assume a random polarity, there are still two ways in which the resulting pattern follows an acceptable encoding. In the case $q = 0.1$, we get an expected fraction of correct data of $p(q = 0.1) = 0.77$. Indeed, evaluating `Our-score` on the initial data set we get a value of ~ 0.77. During the training phase, we did not add an amplitude to the exponent of the Boltzmann weight: this means that we were not making any denoising and the RBM was simply reproducing the original distribution of the data, including the random fluctuations. This is why the plot of `Our_score` stabilizes at 0.77.

Visual representation of the weights

By picturing the weights of the RBM after training, we get a glimpse of the pattern understood by the machine.

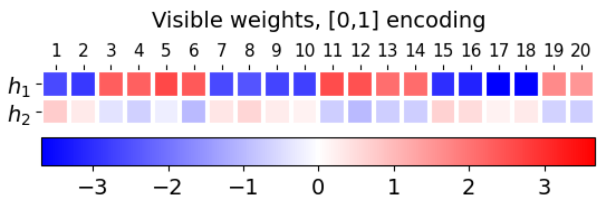
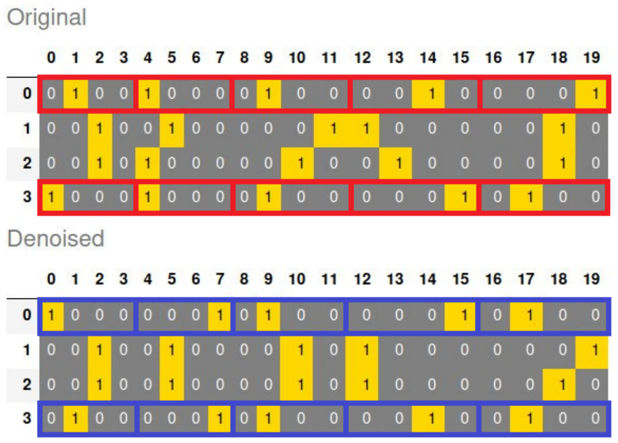


Figure 4. Representation of the weights in the first data set. Each row corresponds to a hidden variable and the color of each cell represents the corresponding weight.

From the alternation of the red and blue cells (see Figure 4, the RBM shows a good understanding of the data pattern. Indeed, it is saying that adjacent nibbles should have alternating behaviour. We see that the weights of the first hidden variable h_1 are systematically higher in magnitude compared to the ones of h_2. This suggests that only one hidden variable would have been enough to properly represent the data using one-hot encoding.

Data denoising

We can use RBMs to denoise the data. This can be easily achieved by amplifying the exponent of the Boltzmann factors when sampling the new data. This increases the energy gap between the configurations, encouraging the machine to pick a state of low energy. In Figure 5 we show the first few rows of the original and denoised data sets.


Original

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	1	0	0	0	0	1
1	0	0	1	0	0	1	0	0	0	0	0	1	1	0	0	0	0	0	1	0
2	0	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0
3	1	0	0	0	1	0	0	0	0	1	0	0	0	0	0	1	0	1	0	0

Denoised

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	1	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	1	0	0
1	0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	0	1
2	0	0	1	0	0	1	0	0	0	0	1	0	1	0	0	0	0	0	1	0
3	0	1	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0

Figure 5. First rows of the original and denoised data sets. The noise (for example in the first and fourth rows of the original data set) is completely deleted.

The value of `Our_score` on the denoised data set is of 1.0, meaning that all data have been correctly denoised.

Differences in data representation

We also trained the RBM using the $[-1, 1]$ representation, to picture how the weights are different from the $[0, 1]$ case. We kept $CD = 20$ steps of Contrastive Divergence and 2 hidden units.

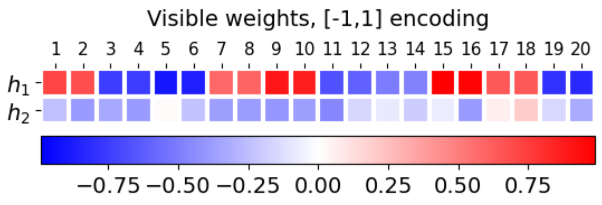


Figure 6. Heatmap of the weights using SPINS $[-1, 1]$.

By looking at the weights in Figure 6, we notice that the weights of one hidden variable are systematically higher in magnitude. Indeed, one of the two hidden units is neglected by the RBM, similarly to what happens in the $[0, 1]$ case.

Making a comparison with Figure 4, we observe that weights are clearly more visible in the former case, thus letting us prefer the $[0, 1]$ representation.

Centering trick

Since data representation influences RBM performances, we tried to implement the "Centering trick" [3]. This should even out the differences in the results from $[0, 1]$ and $[-1, 1]$ representation, while also improving the overall performance. We confronted the log-likelihood and the AAI index in the two cases, but the results we obtained are not a clear improvement of what showed before.

After many runs and attempts we decided to leave the code implemented in the notebook but not to present any evidence here, since the log-likelihood does not improve.

Training without prior knowledge

We also try to study the performances of the RBM on the initial data set without one-hot encoding. This means that the generated data during training can assume any value in the 2^{20}-sized set of possible configurations and that the RBM has to *understand* on its own that it has to implement one-hot encoding. This significantly increases the computational demand required for the partition function calculation. Thus, we can only consider the AAI as estimator in order to get a graphical representation of the learning curve, since the evaluation of the partition function would be unfeasible.

We therefore train the RBM using Adam algorithm with $[0, 1]$ data representation and 20 CD-steps for 100 epochs with 6 hidden units, but the results are not satisfactory. So, in order to get better performances, we changed the training configurations, acting on the mini batch size, the number of CD-steps as well as the number of epochs.

Unfortunately, in the end we were not able to get results comparable with those expected and found with the one-hot encoding function. The best score reached has been of 20% with the function `Our_score` and 1000 epochs. But, the AAI index still fluctuates: this suggests that the training time has to be largely increased in order to get better and better results. It is worth mentioning, however, that even though `Our_score` is a really strict evaluator, the RBM still gets a non zero score value.

UNKNOWN DATA SET

Finally, we apply the best-settings RBM to the second data set of unknown distribution. The only prior knowledge we possess is that the data set is also generated through one-hot encoding, with each datum consisting of $G = 5$ blocks of $A = 6$ bits each.

We train the RBM on this data set using the best configuration found in the previous sections and with the one-hot encoding function adapted to the case of $A = 6$.

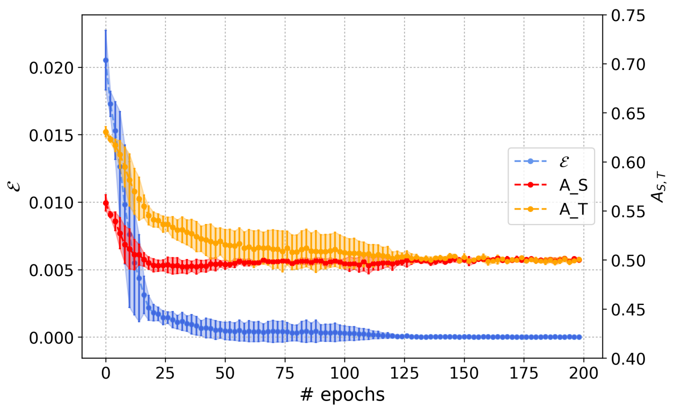


Figure 7. $\mathcal{E}$, A_S and A_T as a function of the epochs when training on the unknown data set.

By looking at Figure 7, we notice that there is more variability in the evolution, compared to what happened in the first data set. This means that the system reaches the stability after a different number of epochs depending on the random seed: for all the simulations we have run it converges after 150 epoches. In order to discover the pattern followed by the data, we also analyse the weights plot, looking for a hidden structure.

From the heatplot of the weights, it is clear that the two hidden variables are performing a division of labour.

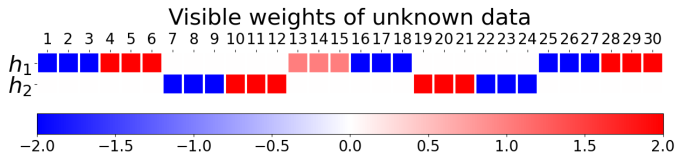


Figure 8. Weights of the RBM when trained on the unknown data.

The first, third and fifth block are controlled by the first hidden unit h_1, whereas the others are controlled by h_2. The pattern behind the data is the alternation of polarity between one block and its second nearest neighbour, with the 1 of the one-hot encoding switching from "left" (first three bits) to "right" (last three bits). We therefore have four possible states, noise notwithstanding: $LLRRL$, $LRRLL$, $RLLRR$, $RRLLR$. The fraction of data in the original data set which followed this pattern was 0.6. Given a probability q of a block to randomly switch its state, we have that the probability for a datum to be correct is:

$$\left[\left(1-\frac{q}{2}\right)^3+\left(\frac{q}{2}\right)^3\right]\cdot\left[\left(1-\frac{q}{2}\right)^2+\left(\frac{q}{2}\right)^2\right] \qquad (7)$$

which leads to the observed probability of 0.6 if $q = 0.2$.

CONCLUSIONS

In this work, we trained an RBM with two hidden variables on two different data sets, testing the ability of the model to detect unknown patterns and perform data denoising. The model effectively captured the pattern encoded when generating the first data set. The same model was then able to reach the same results with a higher number of hidden variables. It also allowed us to understand the structure behind a data set whose structure was unknown.

We attempted to train a more complex model, without explicitly constraining it to only generate data through one-hot encoding, without much success. It could be interesting to study further this possibility.

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

[2] Aurélien Decelle, Cyril Furtlehner, and Beatriz Seoane. Equilibrium and non-equilibrium regimes in the learning of restricted boltzmann machines. *Advances in Neural Information Processing Systems*, 2021.

[3] Matteo Bortoletto. Study of performances for restricted boltzmann machines. 2021.