GROUP 9 – Restricted Boltzmann Machines to Learn Patterns in Biological Datasets

Davide Bacilieri, Lorenzo Barbiero, Guglielmo Bordin, and Alessio Pitteri

(Dated: 3rd April 2023)

In the field of unsupervised learning, Restricted Boltzmann Machines excel in pattern recognition and denoising. This leads to applications everywhere that noise or other artefacts need to be sorted out from data with unknown generating rules, with chief examples in Information Theory or – as for this report – biological datasets. In the work described by the present paper, we built an RBM from the ground up to apply it to a toy problem of pattern recognition in pseudo-amino-acidic sequences. We experimented on the training procedure, fine-tuning the model's parameters and exploring the relevant quantities, like the behaviour of the log-likelihood function that is subject to maximization through gradient descent.

INTRODUCTION

Restricted Boltzmann Machines (RBMs) belong to the class of energy-based generative models. They are *generative* in the sense that they attempt to reproduce the underlying probability distribution that governs the generation of the training data, in order to generate new samples that could have been drawn from the same dataset [1].

One use-case of this kind of models is in biology, with protein sequences [2, 3]: RBMs can efficiently and reliably learn complex and recurring patterns hidden in the sequences of amino acids. Indeed, we worked on a very rudimentary implementation of this concept: our dataset consisted of several short sequences of 1 and 0 bits, which could for example encode information about the alternation of polar and non-polar amino acids. The idea is to filter out the noise and inconsistencies from the sequence, and grasp the underlying pattern by having the RBM generate a "clean" sequence after training. Of course, this has no pretence of actual resemblance to reality; it rather serves as a proof of concept to show the capabilities of the learning model.

The learning framework of RBMs strongly resembles many Ising-like models of statistical physics. Taking the same equations, the problem is reframed as the iterative learning of the parameters of a variational distribution that should approximate the true distribution of the data.

Another thing that we borrow from physics is the concept of *hidden variables*. In Ising-like models, one usually performs a Hubbard–Stratonovich transformation to avoid having to deal with the complex quadratic interactions between spins, by shifting to the description of a system of non-interacting spins immersed in a Gaussian field [4]. Similarly, in the context of Boltzmann learning, we can simplify the complex relationships between the variables in the training data by making them interact in a "controlled" manner with an additional layer of fictitious variables.

METHODS

We will borrow the notation from the 2019 review on Machine Learning by Mehta et al. [1]. Like we have said, Restricted Boltzmann Machines are trained with the goal of best approximating a joint probability distribution p of the visible variables $\boldsymbol{v}$ and the hidden variables $\boldsymbol{h}$, which, carrying on the analogy with statistical physics models, is written as a Boltzmann distribution

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{e^{-E(\boldsymbol{v}, \boldsymbol{h})}}{Z}, \tag{1}$$

where the energy function $E(\boldsymbol{v}, \boldsymbol{h})$ takes the form

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i=1}^{N} a_i v_i - \sum_{\mu=1}^{M} b_\mu h_\mu - \sum_{i=1}^{N} \sum_{\mu=1}^{M} W_{i\mu} v_i h_\mu. \tag{2}$$

We will refer to the variables $W_{i\mu}$ as *weights*, and to the coefficients a_i and b_μ as *biases*. These are the parameters to learn, and we will collectively denote them with $\boldsymbol{\vartheta}$.

The actual training, that is, the iterative approximation of the true p with a parametrized $p_{\boldsymbol{\vartheta}}$, is performed through the Maximum Likelihood Estimation procedure. This involves the maximization – or minimization of the negative – of the *log-likelihood* $\mathcal{L}(\boldsymbol{\vartheta})$ of the model:

$$\mathcal{L}(\boldsymbol{\vartheta}) = -\langle E(\boldsymbol{\vartheta}) \rangle_{\text{data}} - \log(Z(\boldsymbol{\vartheta})). \tag{3}$$

In practice, this translates to the application of (stochastic) gradient descent methods to the following set of equations:

$$\frac{\partial \mathcal{L}(\boldsymbol{\vartheta})}{\partial W_{i\mu}} = \langle v_i h_\mu \rangle_{\text{data}} - \langle v_i h_\mu \rangle_{\text{model}}, \tag{4}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{\vartheta})}{\partial a_i} = \langle v_i \rangle_{\text{data}} - \langle v_i \rangle_{\text{model}}, \tag{5}$$

$$\frac{\partial \mathcal{L}(\boldsymbol{\vartheta})}{\partial b_\mu} = \langle h_\mu \rangle_{\text{data}} - \langle h_\mu \rangle_{\text{model}}. \tag{6}$$

Here the expectation values with respect to the data are the empirical averages on the training samples, while the expectation values with respect to the model need to be estimated by drawing samples from the parametrized distribution $p_{\boldsymbol{\vartheta}}$.

For the problem at hand, we chose a hidden layer of two units and decided to carry out the gradient descent in mini-batches of 500 samples, using the *ADAM* algorithm

[1]. We also attempted to implement the *RMSProp* algorithm, but ultimately settled on *ADAM* due to its (slightly) more reliable performance.

To calculate the expectation values in Eqs. (4)–(6), we performed three steps of block Gibbs sampling (so, a *three-step Contrastive Divergence*), each step being a forward sample from the visible layer to the hidden layer, plus a backward sample in the opposite direction.

We should now specify that our training data consisted of many sequences of one-hot encoded blocks of units of length B. These blocks were interpreted as encodings for the different amino acids chaining to form a "protein". So, while the hidden layer posed no sampling problems – the h_μs are simple binary variables – the backward sample on the visible layer required some special attention due to the one-hot encoding.

Let us define the possible values of each data block in $\boldsymbol{v}$ as $\boldsymbol{v}^{(k)}$, with $k = 1, \dots, B$ denoting the position of the positive bit. During the backward sampling, we have to clamp the variables in the visible layer to every possible configuration and compute their respective probability. Therefore, a given block has B different probabilities associated to it:

$$p(\boldsymbol{v}^{(k)} \mid \boldsymbol{h}) = \frac{1}{Z} \exp\left[a_k + \sum_{\mu=1}^{M} (b_\mu + W_{k\mu}) h_\mu\right]. \tag{7}$$

The partition function Z is the sium of the B probabilities, so the term with $\boldsymbol{b}$ cancels out. In practice, for each block of B units in $\boldsymbol{v}$, we computed the cumulative probabilities C_k as $\tilde{C}_k/\tilde{C}_B$ with

$$\tilde{C}_k = \sum_{i=1}^{k} \exp\left(a_i + \sum_{\mu=1}^{M} W_{i\mu} h_\mu\right) \tag{8}$$

Then, we generated a random number r, chose ℓ such that $C_{\ell-1} < r < C_\ell$, and replaced the block with $\boldsymbol{v}^{(\ell)}$.

We also explored the possibility to work with "spins" instead of bits, i.e. $+1/-1$ binary variables. Eq. (7) has to be reworked a bit, giving

$$p(\boldsymbol{v}^{(k)} \mid \boldsymbol{h}) = \frac{1}{Z'} \exp\left[-\sum_{j=1}^{B} (-1)^{\delta_{jk}} \left(a_j + \sum_{\mu=1}^{M} W_{j\mu} h_\mu\right)\right], \tag{9}$$

where Z' is the partition function without the terms with $\boldsymbol{b}$ that cancel out. We can use a similar expression as Eq. (8) but with a factor of 2 multiplying the argument of the exponentials. The reason for this becomes clear if we factor out from Eq. (9) the term

$$\exp\left[-\sum_{j=1}^{B} \left(a_j + \sum_{\mu=1}^{M} W_{j\mu} h_\mu\right)\right]. \tag{10}$$

Although the $+1/-1$ representation has undoubtedly a nice connection to Physics notation, the $1/0$ version turned out to be more computationally efficient and conceptually easier to implement. So, in the end we opted for the latter.

As regards the calculation of the partition function Z that enters the expression of the log-likelihood, the $0/1$ format allows the following simplification:

$$\begin{aligned} Z &= \sum_{\{\boldsymbol{v},\boldsymbol{h}\}} e^{-E_{\boldsymbol{\vartheta}}(\boldsymbol{v},\boldsymbol{h})} \\ &= \sum_{\{\boldsymbol{h}\}} e^{B\boldsymbol{b}\cdot\boldsymbol{h}} \prod_{j=0}^{N_B} \sum_{k=j}^{j+B-1} e^{a_k + W_{k\mu} h_\mu}, \end{aligned} \tag{11}$$

where the sums on $\{\boldsymbol{v}, \boldsymbol{h}\}$ and $\{\boldsymbol{h}\}$ denote the sum over all possible configurations of positive and negative bits.

Before moving on to the results discussion, we should mention that we implemented the Adversarial Accuracy Indicator (AAI) from [5] to provide a meaningful rating for our trained model. The task of finding accuracy indicators in the context of generative learning has no definitive solution, unlike in supervised learning. With AAI, the similarity between the original dataset and a generated one is gauged by counting how many times a given data point's nearest neighbour belongs to the same set and how many times it belongs to the other; if indeed the sets came from the same probability distribution, the ratio of nearest neighbours of the same type to the total number of samples should approach $1/2$.

The distance we used to choose the nearest points is a Manhattan-like distance in L dimensions, where L denotes the number of units in each sequence of the dataset. That is, we count how many bits are different between two sequences to quantify their distance. We perform this procedure for the entire dataset divided into mini-batches of only 10 sequences, because there is a high chance that in a larger dataset many samples would find an exact copy of themselves.

RESULTS

We had two datasets at our disposal: one with one-hot encoded blocks of 4 bits and another with 6-bit blocks. The pattern in the 4-bit dataset was known a priori, unlike the pattern in the 6-bit data. Therefore, the plan was to use the first dataset to evaluate the model's performance. If the result was satisfactory, we would have proceeded to analyse the unknown dataset.

The training with the 4-bit dataset went smoothly, and the model correctly guessed the schema. The pattern to learn was the alternance of two kinds of "amino acids". One of the amino acids was identified by either of the two one-hot blocks with the positive bit in the first two positions, while the other two blocks identified the other amino acid. To preserve the physical analogy, we can classify the first type as a general polar amino acid and

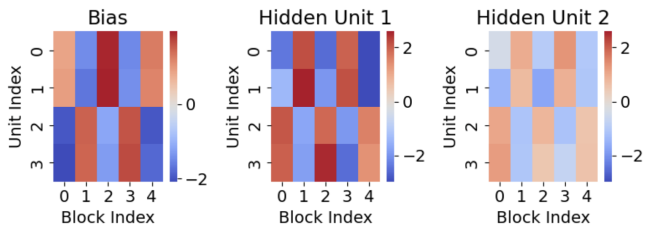


Figure 1. Heatmap depiction of the $\boldsymbol{a}$ (bias), h_1, h_2 matrices for the 4-bit dataset. The block index runs across the one-hot encoded blocks in each row of the dataset, while the unit index denotes each unit within a block.

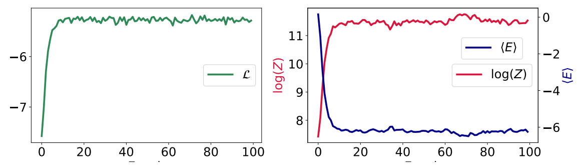
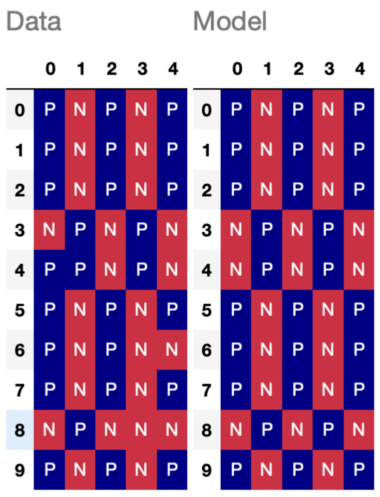


Figure 2. Behaviour of $\mathcal{L}$, $\langle E\rangle$ and $\log(Z)$ as a function of the training epochs. An increase in $\mathcal{L}$ coupled with a decrease in $\log(Z)$ hints to the good quality of the training.

the second type as a non-polar amino-acid. As in the real world, a polar amino acid comes after a non-polar one in a linear sequence.

In Fig. 3, you can see a comparison between a portion of the original dataset and a generated one. The original dataset has some noise that disrupts the polar/non-polar alternation in various places, while the generated sample corrects the imperfections and returns consistent sequences in every line. The denoised sequences are computed using the biases and weights from the last epoch in training, with a single divergent step.


Data

	0	1	2	3	4
0	P	N	P	N	P
1	P	N	P	N	P
2	P	N	P	N	P
3	N	P	N	P	N
4	P	P	N	P	N
5	P	N	P	N	P
6	P	N	P	N	N
7	P	N	P	N	P
8	N	P	N	N	N
9	P	N	P	N	P

Model

	0	1	2	3	4
0	P	N	P	N	P
1	P	N	P	N	P
2	P	N	P	N	P
3	N	P	N	P	N
4	N	P	N	P	N
5	P	N	P	N	P
6	P	N	P	N	P
7	P	N	P	N	P
8	N	P	N	P	N
9	P	N	P	N	P

Figure 3. Excerpt from the original dataset with 4-bit blocks next to one generated by the model. N stands for *non-polar*, P for *polar*.

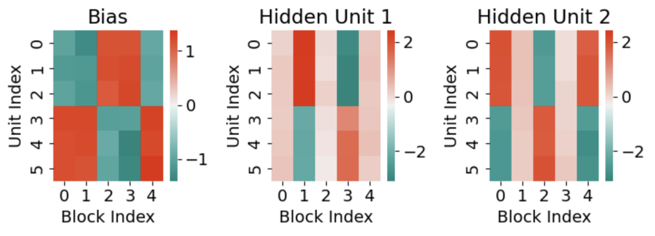


Figure 4. Heatmap depiction of the $\boldsymbol{a}$ (bias), h_1, h_2 matrices for the 6-bit dataset. Refer to Fig. 1 for the notation.

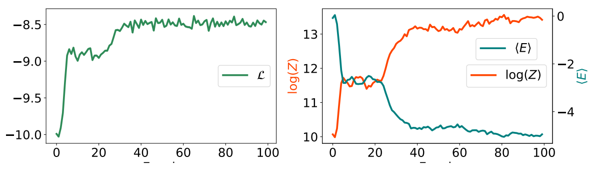


Figure 5. Behaviour of $\mathcal{L}$, $\langle E\rangle$ and $\log(Z)$ as a function of the training epochs.

A graphical representation of the fully trained model is shown in Fig. 1. The alternation of positive and negative values in each of the three quantities – bias and hidden units – is rightfully indicative of the polar/non-polar alternation intrinsic to the data. We have to keep in mind that the blocks are one-hot encoded, so the first two units in a block are essentially encoding the same information, as are the last two; it is not surprising, then, that the heatmaps are split along the middle.

We also introduced a parameter β to rescale the energy and simulate a decrease in temperature, reducing the stochasticity in the generation. This is analogous to the temperature in an Ising-like system; in fact, the Hamiltonian (2) can be mapped to that of a Hopfield model [1] – with the important distinction that the hidden layer is binary instead of Gaussian. We know then that perfect pattern retrieval happens only when the temperature is low enough.

Since the training results for the 4-bit datasets were satisfactory, we moved on to the 6-bit dataset with the unknown generating distribution. We followed the same training procedure, and the resulting model's heatmaps are shown in Fig. 4. Especially in this case, we found the plots shown in Fig. 5 to be of paramount importance to the quality control of the training procedure. It is also interesting to observe a tradeoff mechanism between the maximization of the partition function and the minimization of the mean energy, leading to an overall stable value for $\mathcal{L}$. This is visible after the first twenty epochs in particular: around that time the RBM is usually able to escape from local minima and reach better, and eventually final, states.

Once again, we can clearly observe an underlying pattern in the heatmaps: the blocks $\boldsymbol{v}^{(k)}$ with $k = 1, 2, 3$ appear to identify one type of amino acid, while those

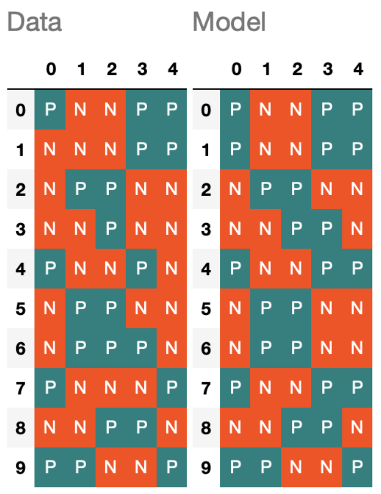


Figure 6. Excerpt from the original dataset with 6-bit blocks next to one generated by the model.

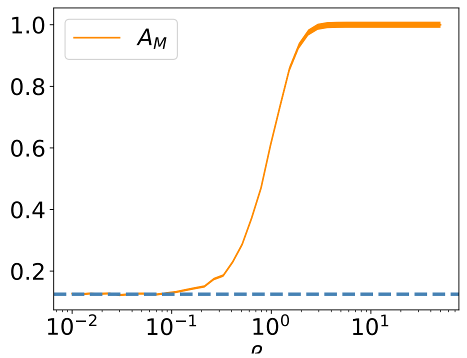


Figure 7. Behaviour of A_M as a function of β. The error bar is given by Poisson variance.

with $k = 4, 5, 6$ identify another type. With the above considerations, we could interpret the data with the same notation N/P as before, but losing the polar/non-polar connotation, since the pattern is no longer a simple alternation. A comparison between a portion of the original dataset and some generated sequences is shown in Fig. 6. The figure clearly demonstrates that, if the training was successful, the underlying probability distribution generates sequences with an alternation of pairs of blocks of the same type.

We also decided to inquire more into the "temperature" parameter β and how changes in its value impact the model's output. To this end, we constructed an accuracy indicator A_M telling us how many sequences in a generated dataset present a pattern that we supposed to be correct, based on the results from the previous training runs.

As shown in Fig. 7, we found a dependence of A_M on β that is quite reasonable. In the low β – "high temperature" – regime, the RBM is just randomly generating bit blocks: therefore, for sequences of 5 blocks like the ones we had, we can expect an accuracy of $4/2^5 = 1/8$ in the 6-bit case, and $2/2^5 = 1/16$ for the 4-bit one. Then, after a certain critical value of β, the RBM suddenly shifts from producing random noise to guessing the correct patterns, and the ratio eventually A_M plateaus to 1.

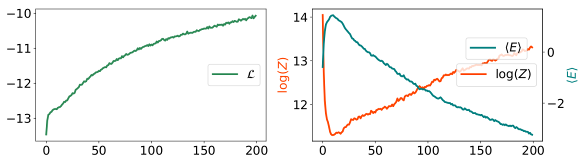


Figure 8. Behaviour of $\mathcal{L}$, $\langle E \rangle$ and $\log(Z)$ as a function of the training epochs in the $B = 6$ binary case.

Finally, one can ask what might happen if we translated the original one-hot encoded data into a straight binary format. In the $B = 4$ case, just 2 bits suffice to encode all the one-hot blocks:

$$\begin{aligned} 0001 &\rightarrow 00 \\ 0010 &\rightarrow 01 \\ 0100 &\rightarrow 10 \\ 1000 &\rightarrow 11 \end{aligned} \tag{12}$$

Instead, 3 bits would be required for the 6-bit dataset conversion, but only 6 out of the 8 possible states would be paired to a one-hot block. The points in favour of this method are a reduced size of the training set, and a much simpler implementation of the sampling function. On the other hand, for block sizes that are not powers of 2 – like for our 6-bit dataset – the presence of spurious binary states that have no one-hot counterpart might lead to wrong pattern recognition. To have a more conclusive answer, further research is needed.

While in the simpler 4-bit case the performance is quite similar to the one-hot encoded version, with good error correction, in the 6-bit case the correction is often diverging from the sequence suggested by the one-hot results.

We might have found a plausible explanation in the binary representation for the 4-bit case. There, the values 00 and 01 represent one amino acid type while 10 and 11 represent the other, so the overall pattern only depends on odd bits. This behaviour does not translate to the 6-bit case, and thus the pattern becomes much harder to learn. In fact, the algorithm does not seem to reach convergence even after doubling the training epoch – even though the accuracy indicator is of the same order of magnitude as in the one-hot encoded version.

AUTHOR CONTRIBUTION

All members of the group have meaningfully contributed to the research, implementation and writing of the report. In particular, the acknowledgment goes to Guglielmo Bordin for the development and implementation of theoretical methods, to Davide Bacilieri and Alessio Pitteri for the coding framework's optimization and to Lorenzo Barbiero for data visualization and presentation.

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

[2] J. Tubiana, S. Cocco, and R. Monasson, Learning protein constitutive motifs from sequence data, eLife **8**, e39397 (2019).

[3] J. Tubiana, S. Cocco, and R. Monasson, Learning Compositional Representations of Interacting Systems with Restricted Boltzmann Machines: Comparative Study of Lattice Proteins, Neural Computation **31**, 1671 (2019), https://direct.mit.edu/neco/article-pdf/31/8/1671/1053381/neco_a_01210.pdf.

[4] J. Hubbard, Calculation of partition functions, Phys. Rev. Lett. **3**, 77 (1959).

[5] A. Yale, S. Dash, R. Dutta, I. Guyon, A. Pavao, and K. P. Bennett, Generation and evaluation of privacy preserving synthetic health data, Neurocomputing **416**, 244 (2020).