Training Restricted Boltzmann Machines

Guillermo Benito, Claudia Lorenzetti, Leon Mengoni, and Filippo Pra Floriani

(Dated: January 9, 2024)

The process of drawing and generating data from an unknown distribution is fundamental in many scientific fields. In past years, unsupervised machine learning techniques have been increasingly investigated for their ability to learn correlations in unlabeled data. This work focuses on Restricted Boltzmann Machines (RBMs), which are used to highlight the underlying patterns of real data and to reproduce them. The main challenge, however, is modelling and training these networks. Throughout this work, different ways of improving a RBM and different methods of measuring its performance are employed on two mock datasets. Overall, we find that simple models work better.

I. INTRODUCTION

In the broad category of unsupervised learning, *generative models* are a potent machine learning tool used to learn patterns in complex data structures and draw new data from unknown probability distributions. These types of algorithms sharply contrast with *discriminative models*—used in supervised learning tasks—which are, on the other hand, constructed to discern differences between separate classes of data. In practice, however, the scope of discriminative models is strongly limited by the availability of labeled data.

Therefore, when dealing with complex, unlabeled, real-world data, generative models provide the best blend of adaptability and versatility, that enable their users to carry out a number of tasks, including discrimination and denoising [1].

In this work, we train an energy-based generative stochastic model called *Restricted Boltzmann Machine* (RBM). Our objective is to learn the underlying patterns in data fraught with random noise, and to generate denoised data. In order to accomplish this, a multitude of hyperparameters, such as the optimization algorithm, the architecture of the machine and the number of steps in the employed Markov Chain Monte Carlo (MCMC) method (Gibbs Sampling), have to be tweaked. As emphasized in [2], however, quantifying the training performance of an RBM is not an easy task: therefore, we simultaneously evaluate several quality indicators, such as the Adversarial Accuracy Indicator (AAI), the model energy and the log-likelihood. We then identify the best model as the one that returns the best values (and the correct behavior) for these quality indicators.

II. METHODS

A. Model

A Restricted Boltzmann Machine is an energy-based generative stochastic neural network that employs a bipartite structure. Namely, it is composed of visible and hidden (latent) units which interact with each other, but not among themselves [1].

Denoting as v_i, $i = 1, ..., N_v$, the visible and as h_j, $j = 1, ..., N_h$, the hidden discrete binary units ({-1,1} spins or {0,1} bits), the energy function is defined as:

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

where $\mathbf{w}$ is the weight matrix (representing the coupling between visible and hidden units), while $\mathbf{a}$ and $\mathbf{b}$ are the visible and hidden local biases.

For any configuration of $\mathbf{v}$ and $\mathbf{h}$, the probability distribution for the RBM state is given by the temperature-dependent Boltzmann distribution:

$$p(\mathbf{v},\mathbf{h}) = \frac{e^{-\beta E(\mathbf{v},\mathbf{h})}}{Z} \tag{2}$$

where the partition function is defined as

$$Z = \sum_{\{\mathbf{v},\mathbf{h}\}} e^{-\beta E(\mathbf{v},\mathbf{h})} \tag{3}$$

and where $\beta = \frac{1}{T}$ acts as an amplifier. In practice, a high β (low T) spikes the probability of the states with lower energy, thus making them even more probable, and allows an efficient denoising, while a low β (high T)—such as β=1—preserves the original probabilities, therefore keeping the inherent noise. In practice, this translates into keeping β fixed at a low value during training, while it can be modified to highlight different behaviors in the testing phase.

B. Data

In this work, we analyze two datasets. The first type of data sample is a 30-bit vector divided in five 6-bit blocks, composed of alternating strings of four 1's and two 0's.

	Block 0	Block 1	Block 2	Block 3	Block 4
$v^{(d)}$	001111	111100	001111	111100	001111
$v^{(d')}$	111100	001111	111100	001111	111100

TABLE I. First dataset

These samples are generated with some noise, so that every bit has a 10% probability of flipping. The machine is then trained with the aim of replicating this pattern and denoising the data, i.e. we use this first dataset to test only the denoising capabilities of our machine.

The second type of data sample represents a one-hot encoded chain of 5 amino acids, picked from a fictional pool of 4 amino acids. Every sample is, therefore, a 20-bit vector divided in five 4-bit blocks. When the 1 bit is positioned in the first half of the 4-bit block, we classify the amino acid as *non-polar* (N); viceversa, if the 1 bit is in the second half of the 4-bit block, it is classified as *polar* (P). Each chain is, ultimately, an alternating string of polar and non-polar amino acids.

	Block 0	Block 1	Block 2	Block 3	Block 4	Type
$v^{(d)}$	0100	0010	1000	0010	0100	NPNPN
$v^{(d')}$	0010	1000	0001	0100	0001	PNPNP

TABLE II. Second dataset

This second dataset is also generated with noise: single bits do not flip, but every N or P block has a 10% probability of being classified wrongly, therefore not preserving the alternating structure of the chain. The aim of the machine, here, is to generate the correct amino acid chains, by maintaining the one-hot encoded structure.

C. Training

The training of an RBM aims at tuning the weights and biases so that eq.(2) resembles the original data distribution. This is accomplished by maximizing the log-likelihood (LL) function:

$$\begin{aligned}\mathcal{L}(\mathbf{w},\mathbf{a},\mathbf{b}|D) &= \langle \ln p(\mathbf{v}) \rangle_D \\ &= \frac{1}{N}\sum_{d=1}^{N} \ln \sum_{\{\mathbf{h}\}} e^{-\beta E(\mathbf{v}^{(d)},\mathbf{h})} - \ln Z\end{aligned} \tag{4}$$

where

$$p(\mathbf{v}) = \sum_{\{\mathbf{h}\}} p(\mathbf{v},\mathbf{h}) \tag{5}$$

is the marginalized probability distribution, and $D = \{v^{(1)}, v^{(2)}, ..., v^{(N)}\}$ is the training dataset.

The layer of visible units generates fantasy data ('particles') by using an approximate Gibbs Sampling technique called Constrastive Divergence (CD-n), in order to speed up the process. The cost of this optimization is not drawing data from the true model distribution [1].

In all training procedures, the amplitude is fixed at $\beta = 1$.

We now list the hyperparameters used to test different configurations of the RBM.

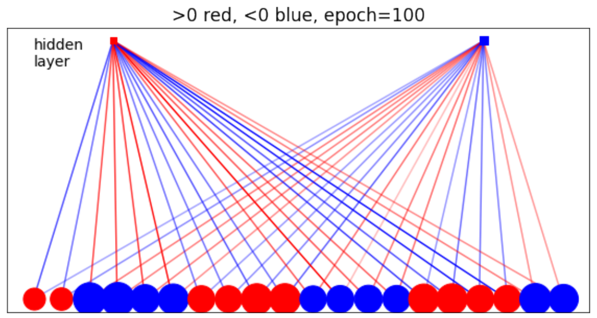


FIG. 1. Graph example of a trained RBM, two hidden units and CD-1

Optimization algorithm: we compare three Stochastic Gradient Descent (SGD) algorithms, which are used in training over minibatches of size 500. The three algorithms are "Vanilla" (standard SGD), RMSprop and ADAM.
Number of hidden units (H): in principle one may expect to capture more correlations in the data if the number of hidden units is high. However, the issue of overfitting and the computational cost must also be considered. Ultimately, we lean towards few hidden units.
Number of CD steps: increasing the number of CD steps may result in generating data that is better sampled from the true model distribution, although overfitting must again be considered.
Centering trick: as presented in [3], we employ this method that consists in shifting the visible and hidden variables by their mean, calculated for every minibatch.

Finally, for the second dataset, we impose a one-hot encoding in the backward step of contrastive divergence. In practice, we create data by blocks, where the probability of each block (and each amino acid) is given by its Boltzmann weight (eq.(2)). Since we want to generate one-hot encoded data, by "helping out" the machine we drastically cut the training time; otherwise it would have to autonomously learn that the data are one-hot encoded, and this would require a larger number of training epochs.

D. Quality Indicators

In order to assess the learning performance, we monitor a series of observables that quantify the quality of the generated samples.

PN control: we check if, for any generated sample, two consecutive blocks are of different kind (polar and non-polar). With this indicator we check if the drawn data are denoised properly.
Adversarial Accuracy Indicator: this indicator measures how similar the original and generated sets are. It is defined as:

$$\mathcal{E}_{AAI} = \left(\frac{1}{2} - \mathcal{A}_S\right)^2 + \left(\frac{1}{2} - \mathcal{A}_T\right)^2 \tag{6}$$

where $\mathcal{A}_S$ and $\mathcal{A}_T$ quantify the similarity of a dataset with the original and generated datasets, as defined in [2]. If $\mathcal{E}_{AAI} = 0$, the two datasets are indistinguishable.

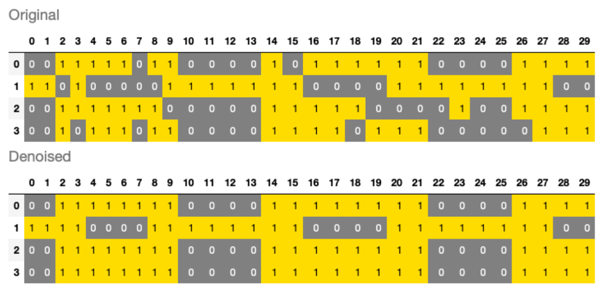
Log-likelihood: eq.(4) represents the probability of generating our data using a particular generative model [1]. If the training is progressing, the log-likelihood behaves as an increasing monotonic function through the epochs until it stabilizes [2].

Energy: as explained in [2], we expect E_{RBM}, the energy calculated over the generated data, to start larger and then converge to E_D, the energy of the original dataset, after several training epochs.

III. RESULTS

For all investigated model configurations, we use the bit representation $\{0, 1\}$.

For the first dataset we found that, by fixing $\beta = 40$ during testing, we can denoise the data perfectly and replicate the underlying pattern, as shown in figure 2.



Original

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	0	1	1	1	1	1	0	1	1	0	0	0	0	1	0	1	1	1	1	1	1	0	0	0	0	1	1	1	1
1	1	1	0	1	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0
2	0	0	1	1	1	1	1	1	1	0	0	0	0	0	1	1	1	1	1	0	0	0	0	1	0	0	1	1	1	1
3	0	0	1	0	1	1	1	0	1	1	0	0	0	0	1	1	1	1	0	1	1	1	0	0	0	0	0	1	1	1

Denoised

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1
1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0
2	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1
3	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	1

FIG. 2. Original and denoised data

For the second dataset, we verify that by not imposing the one-hot encoding of the generated data, the machine is not able to learn correctly.

In order to choose the best model, we implement a "grid search" over the hyperparameters (centering trick excluded) and evaluate $\mathcal{E}_{AAI}$ for each one, with the results collected in table III.

In general, all values of $\mathcal{A}_S$ are close to 0.5 and the error is mainly due to $\mathcal{A}_T$. The values of $\mathcal{A}_T$ are, in fact, slightly bigger than 0.5. As said in [2], this means that the generated samples are more similar among them than to the original data.

The best architecture found for the RBM implements "Vanilla" as an optimization algorithm, with 2 hidden layers and a single step of contrastive divergence. The $\mathcal{E}_{AAI}$ is $4.1 \cdot 10^{-7}$.

$\mathcal{E}_{AAI}$	Hidden layers	CD				
		1	**2**	**5**	**8**	**10**
"Vanilla"	**2**	4e-07	4e-07	5e-07	1e-06	2e-06
	4	9e-07	3e-06	4e-06	4e-06	5e-06
	6	9e-06	5e-06	8e-06	7e-06	8e-06
	10	7e-06	3e-06	8e-07	5e-06	3e-06
RMSprop	**2**	4e-02	4e-02	4e-02	4e-02	4e-02
	4	3e-02	3e-02	3e-02	3e-02	3e-02
	6	3e-03	3e-02	3e-02	3e-02	4e-02
	10	2e-02	2e-02	2e-02	3e-02	2e-02
ADAM	**2**	4e-02	4e-02	4e-02	4e-02	4e-02
	4	3e-02	3e-02	3e-02	3e-02	3e-02
	6	5e-02	5e-02	5e-02	2e-02	2e-02
	10	4e-02	4e-02	4e-02	4e-02	4e-02

TABLE III. Values of the Adversial Accuracy Indicator $\mathcal{E}_{AAI}$ for different models. The lowest one is marked in yellow.

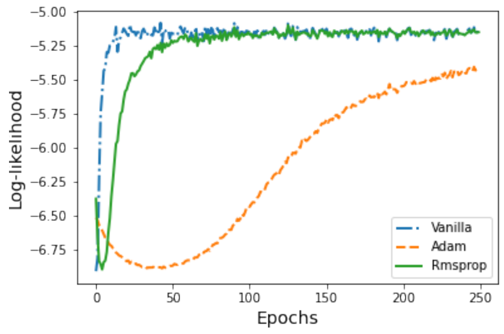


FIG. 3. The log-likelihood behaviour for each algorithm, using the best model according to the Adversial Accuracy Indicator (2 hidden units, CD-1).

Once we define the best architecture, we evaluate the log-likelihood as a function of the training epoch for every optimization algorithm. The graph 3 shows how, after an initial fluctuation, the log-likelihoods increase monotonically. By comparing the three algorithms, we notice that Adam requires more epochs to reach stability, while it is clear that "Vanilla" converges faster.

Now, having chosen the best algorithm—"Vanilla"—We evaluate the log-likelihood $\mathcal{L}_D$ of the original data and compare it to $\mathcal{L}_{RBM}$, of the generated data. We then do the same thing for the energy. In this way, we are able to check if the two functions behave similarly and if the increase of the likelihood in the graph (figure 5) is due to overfitting. The log-likelihood calculated on both the original and generated data converge quickly to the same value. Analogously, E_{RBM} starts slightly higher, but then quickly converges to E_D, as learning progresses.

Regarding the centering trick, we evaluate the Adversarial Accuracy Indicator for the best configuration. The $\mathcal{E}_{AAI}$ is $5.7 \cdot 10^{-7}$. It is slightly greater than the one obtained previously. For our specific dataset and RBM

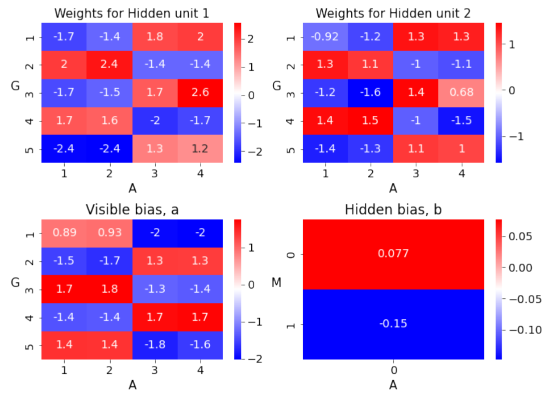


FIG. 4. Heatmap of weights and biases of the best configuration (2 hidden unit and CD-1)

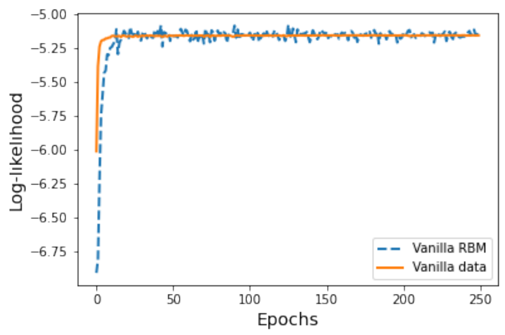


FIG. 5. Comparison between $\mathcal{L}_D$ and $\mathcal{L}_{RBM}$ using the "Vanilla" SGD algorithm for 2 hidden units and CD-1

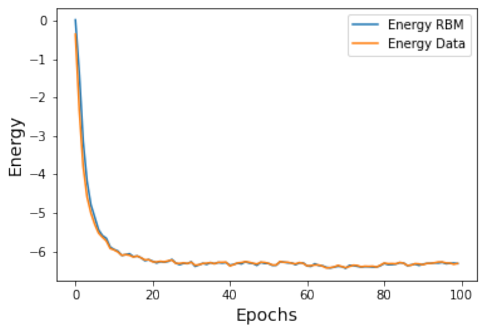


FIG. 6. Comparison between E_D and E_{RBM} using the "Vanilla" SGD algorithm for 2 hidden units and CD-1

configuration, the centering trick does not provide any improvements.

Finally, in order to understand how the RBM learns correlations, we plot a heatmap (figure 4) showing the weights and biases of the best model. Both hidden units behave in the same way, recognising the weights as positive or negative for each 4-bit block.

IV. CONCLUSIONS

Firstly, we analyze a structured dataset beset by noise, with the purpose of denoising it and highlighting its underlying patterns. The model achieves this task when trained with a Boltzmann distribution at low temperature, with $\beta = 40$.

Successively, we train the RBM and evaluate its performance in a high temperature regime, with $\beta = 1$. We test different model configurations, by modifying the optimization algorithm, the number of hidden units and the number of contrastive divergence steps. To verify if the models are trained correctly, we evaluate a numeric indicator, the Adversarial Accuracy Indicator, for all the architectures. The results show that the machine behaves better when the configuration is simpler, i.e. few hidden units and few contrastive divergence steps. For our specific dataset—amino acid chains—, which is not overly complex, this behavior may describe a typical case of overfitting.

Afterwards, we check the dynamics by evaluating the energy and log-likelihood of our models as a function of the training epochs. We find that the training process occurred successfully, resulting in a converging energy function which converges and an increasing monotonic and saturating log-likelihood.

Finally, we investigate an innovative approach, the centering trick. This technique should allow us to improve performance, according to [3], but in our case there is no visible improvement.

[1] P. Mehta, M. Bukov, C.-H. Wang, A. G. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, Physics Reports **810**, 1 (2019), a high-bias, low-variance introduction to Machine Learning for physicists.
[2] A. Decelle, C. Furtlehner, and B. Seoane, Journal of Statistical Mechanics: Theory and Experiment **2022**, 114009 (2022).
[3] M. Bortoletto, *Study of performances for Restricted Boltzmann Machines*, Master's thesis, University of Padua (2021).