Group19 – Application Of RBM Model In Noise Reduction Of Proteins Sequence

Ehsan Esamishafigh, Abbas Zal, Sepideh Ghaemifar, and Poornima Amali Wickramasinghe
(Dated: April 2, 2023)

Most of the Machine learning models that are commonly known known (e.g. linear and logistic regression, ensemble models, and supervised neural networks) are discriminative – they are designed to perceive differences between groups or categories of data. Restricted Boltzmann Machines (RBMs) belong to a different class of machine learning models called generative models. They exploit the maximum entropy principle to estimate the distribution from which the training data comes from. In this way the trained model is capable of producing new "fantasy" data. If the training data has been affected by some noise, the fantasy data produced by the model can be interpreted as denoised data. In this work we used a RBM to denoise a binary dataset belonging to a protein sequence. We test two kind of activation functions for visible layers of the RBM model, namely cell-by-cell activation function and structure preserving activation function. We observe that the latter performs better in denoising the original data. We also test the effect of changing Contrastive Divergence cycles (CDs). Our results show that increasing the number of CDs does not lead to meaningful changes on performance of the model. We Also test the resilience of the RBM model against data affected by high amount of noise. The result show that even in high noise RBM is capable of reducing the noise of training data.

INTRODUCTION

The Restricted Boltzmann Machines (RBMs) are a class of generative models in machine learning in which the goal is to figure out the distribution from which the data has been drawn out. They utilize the approach developed by Jaynes to infer the underling distribution from the provided data.
An RBM model consists of a visible and a hidden layer(fig.1). The visible layer is simply the data provided for training. The correlations between different data points can be modeled as interactions between the different data points. One can draw an analogy between the described model and a physical system such as the Ising model, and write down a generalized energy like:

$$E(x) = -\sum_i a_i x_i - 1/2 \sum_{ij} J_{ij} x_i x_j \qquad (1)$$

Where J_{ij} represents interactions between data point i , j and a_i is a local field at point i. Inspired by the Hopfield model the data points are decoupled from each other by introducing latent variables h_μ which interact with data point i via a weight $W_{i\mu}$. This way the energy can be rewritten as:

$$E(v,h) = -\sum_i a_i(v_i) - \sum_\mu b_\mu(h_\mu) - \sum_i W_{i\mu} v_i h_\mu \qquad (2)$$

If $a_i(v_i)$ and $b_\mu(h_\mu) \in (0,1)$ is binary, they will be $a_i v_i, b_\mu h_\mu$ respectively. The latent variables are referred to as hidden units in RBM model.
In accordance with maximum entropy principle (MaxEnt) the probability of the system taking v , h state is assumed to be in a exponential form like:

$$p(v,h) = \frac{e^{-E(v,h)}}{Z'} \qquad (3)$$

Where Z' is a normalization. p can be considered as the distribution from which the training data is drawn out.

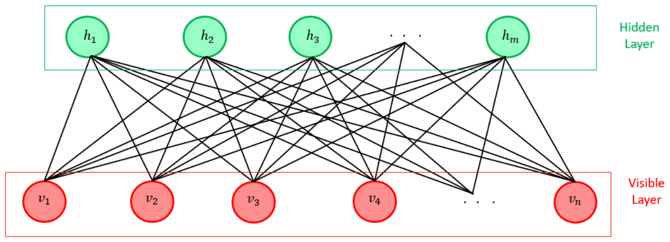


FIG. 1. Restricted Boltzmann Machine (RBM) with connections between visible and hidden layers that interact with each other through this term ($W_{i\mu} v_i h_\mu$) of Eq.(2). Importantly, there are no interactions between visible units themselves or hidden units themselves.

The goal of training a RBM model is to maximize the likelihood of generating the training data using the distribution p which can be characterized by the log-likelihood function:

$$\mathcal{L}(\{\theta_i\}) = <log(P_\theta(x))>_{data} = -E<(x;\{\theta_i\})>_{data} - logZ(\theta_i) \qquad (4)$$

Where θ_i represents the parameters: $a_i, b_\mu, W_{i\mu}$ to be trained.
In practice to find the maximum of $\mathcal{L}$, SGD algorithm is used. From Eq.(2) the gradients of $\mathcal{L}$ with respect to the parameters are calculated as:

$$\frac{\partial \mathcal{L}(\{w_{i\mu}, a_i, b_\mu\})}{\partial w_{i\mu}} =< v_i h_\mu >_{data} - < v_i h_\mu >_{model} \qquad (5)$$

$$\frac{\partial \mathcal{L}(\{w_{i\mu}, a_i, b_\mu\})}{\partial w_{i\mu}} =< v_i >_{data} - < v_i >_{model} \qquad (6)$$

$$\frac{\partial \mathcal{L}(\{w_{i\mu}, a_i, b_\mu\})}{\partial w_{i\mu}} =< h_\mu >_{data} - < h_\mu >_{model} \qquad (7)$$

Where the average over data can be directly calculated from the training data provided while the averages over model should be computed over fantasy data drwan out of the p. In order to sample from p , Markov Chain Monte Carlo (MCMC) method known as Gibbs sampling is used. The idea behind (block) Gibbs sampling is to iteratively sample from the conditional distributions $h_{t+1} \sim p(h|V_t)$ and $v_{t+1} \sim p(V|h_{t+1})$ (fig.2). Since the units are conditionally independent, each step of this iteration can be performed by simply drawing random numbers. The samples are guaranteed to converge to the equilibrium distribution of the model in the limit that $t \to \infty$. At the end of the Gibbs sampling procedure, one ends up with a minibatch of samples (fantasy particles)[1].

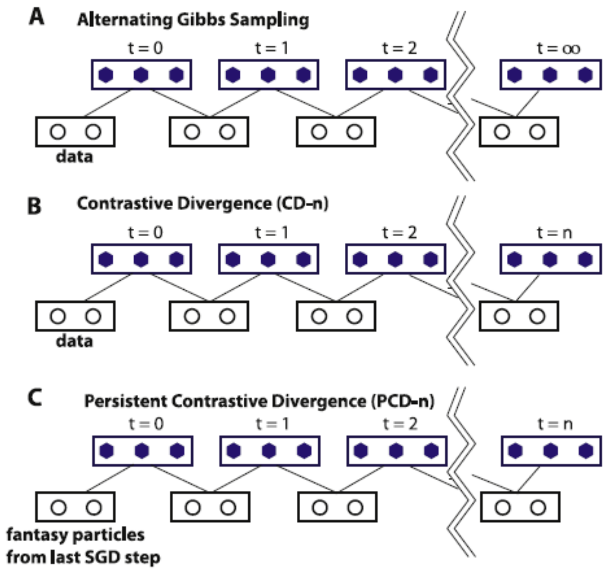


FIG. 2.

METHODS

We applied a RBM model to denoised a sequence of protein data encoded with one-hot encoding technique. The one-hot encoding is a technique to represent the protein sequence as a matrix and encode proteins numerically. We would have a $N \times 20$ matrix, where N represent the number of sequences and 20 is the number of main aminoacid types. For each sequence of proteins, our vector has a '1' to represent the certain type of amino acid while the rest positions will be as '0's.[2][3]
In our project we are using Bernoulli layers for both hidden and visible layer $\in (0, 1)$. By assuming Bernoulli layers Eq.(2) becomes:

$$E(v,h) = -\sum_i a_i v_i - \sum_\mu b_\mu h_\mu - \sum_i W_{i\mu} v_i h_\mu \qquad (8)$$

here a_i and b_μ are visible and hidden local fields respectively.

Using Eq.(8) we compute the average energy over data and the partition function to obtain the log-likelihood function. The corresponding functions in the note book are:

```
def energy(v_data , h_data , w , a , b )
def partition(w , a , b)
```

We track the log-likelihood function to make sure the training converges.
Starting from the visible data we can construct the hidden layers using the cell by cell activation function.The corresponding function in the note book is called simple-activate.

```
def simple_activate(v_in,wei,bias,DE,info=False)
```

The value of each hidden unit is decided using the relation:

$$p(\nu_i = 1 \mid h) = \sigma(a_i + \sum_\mu W_{i\mu} h_\mu) \qquad (9)$$

Where $\sigma(z) = 1/(1 + e^{-z})$ is the sigmoid function.
Next, To generate fantasy visible units from hidden layers we used another type of activation which we call activate-block.

```
def activate_block(h,wei_block,a_block, DE)
```

We assign a probability to each allowed structure of block in one-hot encoding model. The probability are given by normalized Boltzmann factors:

$$p_\gamma = \frac{B_\gamma}{\sum_{\gamma'} B_{\gamma'}} \qquad (10)$$

where γ represents the block structure and:

$$B_\gamma = e^{\sum_i \phi_i \nu_i^\gamma}, \phi_i = \sum_\mu W_{i\mu} h_\mu + a_i \qquad (11)$$

Using activate-block function makes training more efficient, as it is equivalent to give the training some information about the allowed structures that each block can take.You can find a schematic explanation in fig.3.
By repeating the activation cycle described for n times one can implement a CD-n.
After updating the visible and hidden layers we compute the gradient of log-likelihood according to Eq(5,6,7). Next we feed the gradients into SGD algorithm in order to update the weight and local fields. We tested both Vanila and ADAM algorithm.

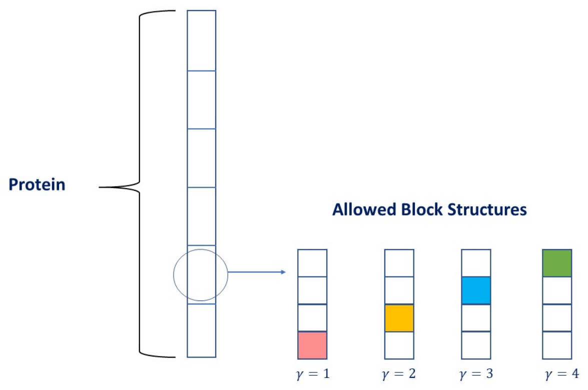


FIG. 3. the schematic explanation of activate-block function

RESULTS

We visualize our trained RBM model in several ways. By looking at fig.4 one can get some rough information about the outcome of the model. For example, in this particular trained model, we can see that one of the units in the hidden layer have been dropped out during the training as the weights entering it are almost zero. This means that the training has automatically recognized, 2 hidden units are enough for generating this type of data. Moreover, the big circles within the visible layer show that the fantasy data corresponding to these units are certainly equal to zero as the weights entering to these units are almost zero.

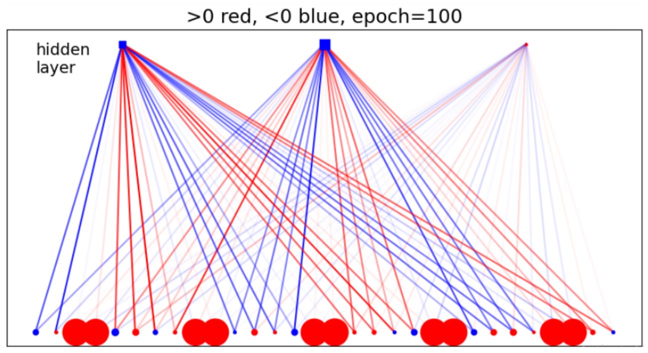
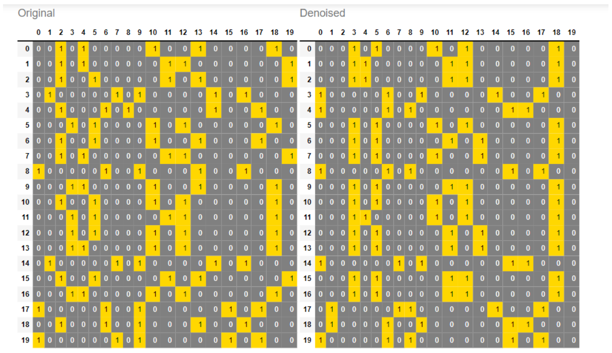


FIG. 4. A view of the trained model, with 30 units in visible layer and 3 units in the hidden layer. The circles and squares represent the local fields in the visible and hidden layer respectively. The lines represent the weights and the red and blue colors represent positive and negative values respectively

We also visualize directly the original data and the denised data (fantasy data producesd by the model) in a tables like fig.5 .

For most of the project we focused on a data of 10000 rows, with each row consisting of 20 data points, encoded with one-hot encoding method. In this type of encoding, each row consists of 5 blocks each with 4 visible units (See fig.5) at each block there is only one unit equal to 1 and the rest are 0. The position of the unit equal to one follows a pattern: it alternates between the first and second half of the block, as we move from one block to the neighboring block. It is easy to show that there is a total of 64 allowed configurations for each row in the data. However, our original data is also affected by some noise, and therefore the original data contains rows which do not belong to any of the 64 allowed patterns with one-hot encoding method. The goal of the training is to produce a denoised data, which contains fewer rows with deformed patterns.

In order to evaluate the performance of the trained model, we introduce a new quantity which we call the percentage of deformed rows (PDR). It is equal to the ratio of the rows with deformed patterns to the total number of rows in the data. We calculate the PDR ratio for both the original and the denoised data produced with our model, and compare their values. If PDR of the denoised data is less than the original data, it mean the model is performing well.



Original

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

FIG. 5. the original and denoised data.

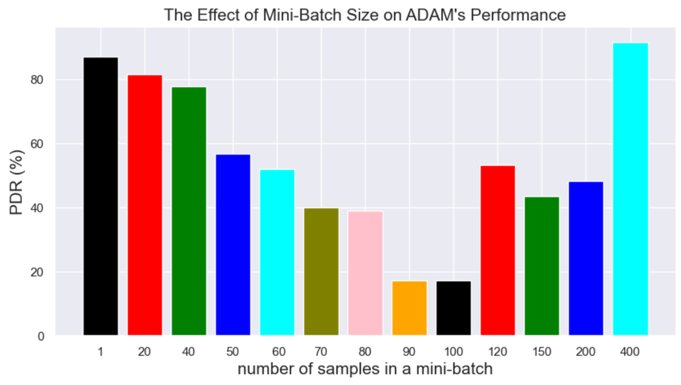


FIG. 6. For these results, the cell-by-cell activation function is used.

We train the model over several epochs. We can make sure that the training has converged, by looking at the plot of the log-likelihood function against the epochs. An example of such a graph is shown in fig.6 . You can see the curve reaches a maximum and becomes flat from some point on.

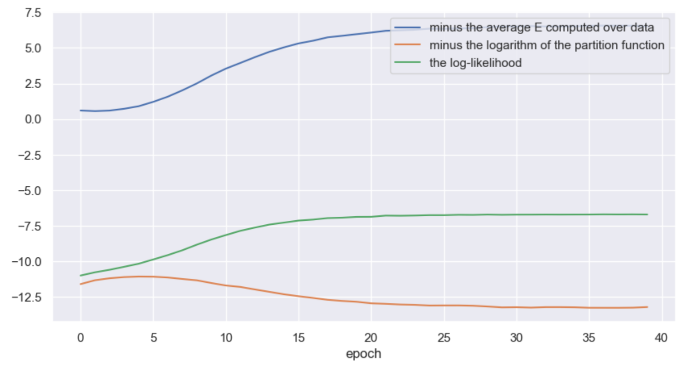


FIG. 7. The change of log-likelihood functions over epochs. The blue and orange curve represent the two terms present in the expression for the log-likelihood in Eq.4 .

The size of the mini-batch for the ADAM algorithm in the training proved to be important. In fig.7 , you can see the effect of variyng the mini-batch size between 1 and 400 for a model trained with cell-by-cell activation function. As we see choosing the size to be around 100 leads to the best performance. It seems that the randomness in the algorithm by choosing a relatively small mini-batch size, helps the training to jump out of the local minimas and achieve better PDR scores. We tested both the cell-by-cell and the structure preserving activation function for the training. Another parameter is the number of CD cycles which we apply during the activation. The results are shown in fig.8a and 8b. The original data by which the model was trained had a PDR of 22.6%. As you can see, both activation functions reduce the PDR, but the performance of structure preserving activation function is better by 2 orders of magnitude. On the other hand, the number of CDs performed does not change the result considerably. The reason behind the success of the structure preserving function is clear. By using this function, we are feeding some extra information about the allowed patterns, into the training while with cell-by-cell activation, the model has to learn this information from the data by itself. We also tested the resilience of the RBM model against high noises. In fig.9, we compare the PDR value of the original data and the fantasy data produced by the model, for different values of the noise probability q. We see that even in a high noise probability like 0.7 the model is capable to decrease the PDR value considerably.

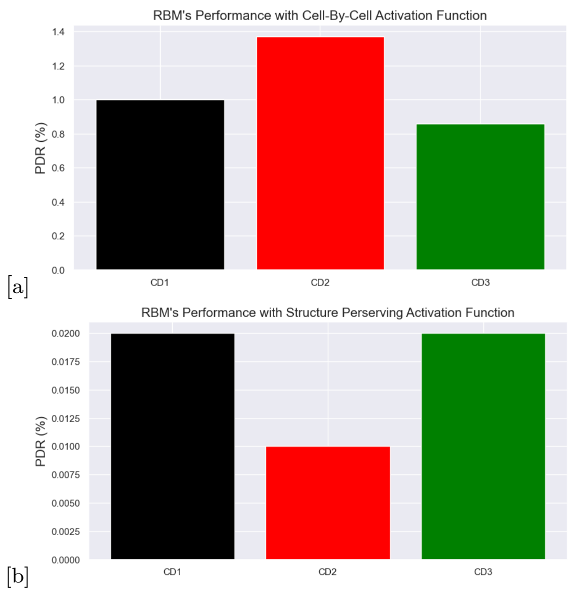


FIG. 8.

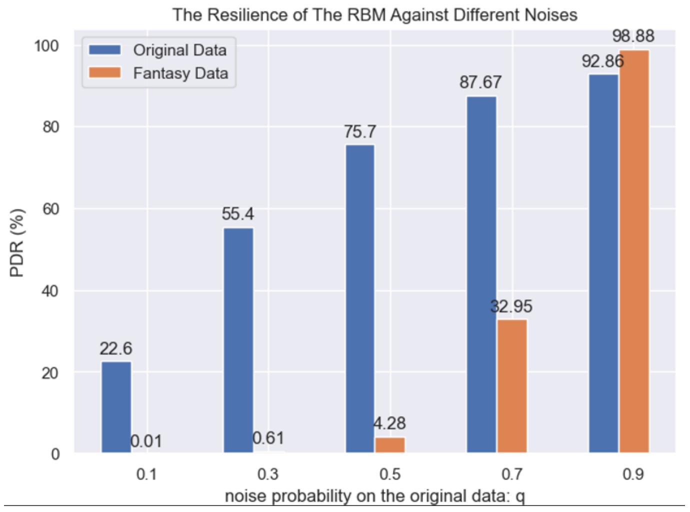


FIG. 9. For these results, the structure preserving activation function is used and the number of CD cycles is set to 2 .

CONCLUSIONS

We applied our RBM model on a binary dataset of protein sequence. We introduced a quantity called percentage of the deformed rows (PDR) by which we could evaluate the performance of our trained model. Our models applied on a dataset of 10000 rows with a PDR of 22.6% were able to produce fantasy data with much less PDR. We saw that using the structure preserving activation function enhances the performance of the model considerably, as the fantasy data produced using this method have a PDR of order 0.01% while this value for a model which uses cell-by-cell activation is of order 1%. We also saw that increasing the number of contrastive divergence cycles does not produce a meaningful effect on the performance of the RBM model.

[1] Pankaj Mehta, Marin Bukov, Ching-Hao Wang, Alexandre G.R. Day, Clint Richardson, Charles K. Fisher, David J. Schwab, A high-bias, low-variance introduction to Machine Learning for physicists, Physics Reports, Volume 810, 2019, Pages 1-124, ISSN 0370-1573.

[2] Li G, Du X, Li X, Zou L, Zhang G, Wu Z. 2021. Prediction of DNA binding proteins using local features and long-term dependencies with primary sequences based on deep learning. PeerJ 9:e11262.

[3] Luo Y, Jiang J, Zhu J, Huang Q, Li W, Wang Y and Gao Y (2022) A Caps-Ubi Model for Protein Ubiquitination Site Prediction. Front. Plant Sci. 13:884903. doi: 10.3389/fpls.2022.884903