Group 2313 – Restricted Boltzmann machines as a Generative model for generic protein structure

Marco Dall'Ara 2091251, Giulio Albertin 2097420, Joan Verguizas I Moliner 2085821, and Max Theodor Peter 2089859

(Dated: April 2, 2023)

We investigate modern optimization techniques for Restricted Boltzmann Machines and their effect on model quality and computation on generic linear proteins consisting of polar and non-polar amino acids sequences. First, the number of hidden layers can be reduced to only one without any loss of generality, reducing computation considerably. Second, implementing Adam compared to Vanilla-SGD shows better stability but slower convergence. Third, increasing the number of Contrastive Divergence steps and performing the Centering Trick allows a faster convergence and performance at the expenses of more computational time.

1. INTRODUCTION

Data allows researchers to analyze and determine the cause of problems in both social and natural sciences though often there is a lack of available data to train their models. This is why artificially generated data is often used to increase the amount of data to achieve better results. Models like the Restricted Boltzmann Machine (RBM), an artificial neural network for unsupervised learning tasks, are often used to learn an unknown underlying probability distribution of given data in order to generate them of the same distribution. However, the problem still remains that RBMs are hard to train and it is difficult to estimate if the distribution of the generated data actually reflects the input data.[1]

In this paper, we investigate empirically different optimization methods for an RBM on a simple, one-dimensional problem, a linear chain of amino acids (AC) that are only differentiated by being polar or non-polar. Through this fundamental approach, we hope to describe the behavior of RBMs in practice in a way that our results can optimistically be taken into account for other problems in the future. First, we will provide a comprehensive overview of RBMs, their architecture, and the training algorithm. Then we return back to the task at hand and analyze data of generated protein chains by the RBM and estimate the model's quality with respect to different optimization settings.

The Boltzmann Machine (BM) is inspired by the Ising model, an energy-based model described by Boltzmann distributions. The Ising model is based on the configuration of the spins and their interactions and gives the probability of finding the system in a particular configuration at a given temperature. Analogous, the BM consists of binary neurons (on / off) that are connected with weights w, that describe the interactions between the spins, and the bias a,b corresponds to the intensity of a local field. In a Restricted BM there are no connections between the units in the same layer but each hidden unit is connected to every visible unit and vice versa. They make use of so-called hidden layers consisting of artificial units that are not directly connected to the visible variables, that have the structure of input data. Two vectors of biases are associated with the visible and hidden variables.

The goal of training is to minimize the difference between the actual and the predicted output which corresponds to maximizing the log-likelihood (LL) function. The LL function quantifies the probability of the RBM generating the observed input data. The Boltzmann distribution, which is described in terms of the RBM's energy, is used to determine the joint probability of the visible and hidden units:

$$P(v,h) = 1/Z * e^{-E(v,h)} \tag{1}$$

where $E(v,h)$ is the energy of the RBM, v and h are the vectors of the visible and hidden unit activations and Z is a normalization constant called partition function. The energy is defined as:

$$E(v,h) = -\sum_i a_i v_i - \sum_j b_j h_j - \sum_{i,j} v_i w_{ij} h_j \tag{2}$$

, where a and b are the biases respectively for the visible and hidden layer, and $w_{i,j}$ are the matrix element of the weight matrix that connects each hidden layer with the visible ones. The partition function is instead defined as the sum of exponentials of energies derived from all the possible combinations between visible and hidden variables :

$$Z = \sum_{i,j} e^{-E(v_i,h_j)} \tag{3}$$

.

The Likelihood function is defined as the logarithm of the mean of the Boltzmann factor over the data taken into account:

$$\mathcal{L} = -\langle log(P(v,h)) \rangle_{data} \tag{4}$$

The derivative of the log-likelihood with respect to the parameters is equal to the difference between a function averaged on the original data and the same function averaged on the data generated by the RBM. Maximizing the log-likelihood implies that the original and generated data have the same distribution.

This algorithm starts by presenting the training data to visible units and by computing the activation of the hidden units. The hidden units then generate so-called "fantasy samples" of the visible units (positive step). Next, the process is turned around and the hidden layer generates new data that should be as similar as possible to the original visible units (negative step). This process is also called Contrastive Divergence(CD).

Moving on to the structure of amino-acids input data, every single chain contains five ACs, each one described by 4 bits, represented binary as either $0, 1$ or $-1, 1$. For $0, 1$ polar ACs are described by [1,0,0,0] or [0,1,0,0], and non-polar ACs are described by [0,0,1,0] or [0,0,0,1]. A protein chain forms an alternating polar and non-polar pattern of ACs in real life as it is the energetically most efficient configuration. For each AC we introduced a random noise with a probability of 10 % of changing one bit.

The goal of the training is to reproduce sequences of polar and non-polar molecules and noise fluctuations. Firstly, we work out which bit notation ($0, 1$ or $-1, 1$) visualizes the alternating pattern clearly and how many hidden layers are needed for a proper outcome. Secondly, we implement the gradient ascend with the ADAM algorithm and observe its performance with respect to the Vanilla SGD set as default. Thirdly, we evaluate the number of CD negative steps needed for a satisfactory prediction. Finally, we will use the Centering Trick to make the machine invariant with respect to translation and spin flips. In order to measure the quality of said changes we compute some indicators such as the log-likelihood (LL) and the second momentum error of the energy to observe the relation between the original and generated data.

2. METHODS

In the one hot encoding condition, we allow only 4 configurations of polar and non-polar amino acids, as written above. Then in this configuration, while for the positive step we maintain the activation function with the following forms:

$$p(h_j = 1|v) = \sigma(b_j + \sum_i w_{ji}^T v_i) \tag{5}$$

(where σ is the sigmoid function) , we change the one of the negative step of the CD:

- Calculate the energy of the four allowed configurations $\{E_i | i = 1, 2, 3, 4\}$ with the corresponding parameters calculated in the positive step.
- Find their corresponding Boltzmann probability $\{p_i | i = 1, 2, 3, 4\}$, with $p_i = \dfrac{e^{-E_i}}{Z}$ and $Z = \sum_{i=1}^4 p_i$
- Extract 5 consecutive configurations from the probability distribution just calculated in order to reproduce the chain
- Use that 20-element array as the visible values generated by the negative step of CD.

In the one hot encoding condition, we expect that at most two hidden variables are needed to detect the polar - non-polar pattern. Then we run the code with a different number of hidden variables and we evaluate the performance with an algorithm that counts adjacent amino-acids with the same polarity. In order to vizualize the RBMs generated pattern we are plotting he weights w_{ij} of the visible layer for both $\{0, 1\}$ and $\{-1, 1\}$ bit notation. In addition to the number of errors, we calculate the difference between the average energy over the original data and the average energy over the data created by the RBM and also the LL of the data:

$$\Delta E = \langle E \rangle_{data} - \langle E \rangle_{model} \tag{6}$$

$$\mathcal{L}_{data} = -\langle E \rangle_{data} - \log Z \tag{7}$$

where Z is the partition function over all values of h and visible values considering one hot encoding condition. If a more complex analysis is needed, we check also the error of the second momentum as written in [1]:

$$\epsilon^{(2)} = \frac{2}{N_v(N_v - 1)} \sum_{i<j} (C_{ij}^{RBM} - C_{ij}^{data})^2 \quad , \tag{8}$$

where N_v is the number of visible layers and $C_{ij}^{RBM/data}$ are the covariance matrices. Note that in energy computations we use only one step in the CD.

Another aspect we want to check is if the stability of the algorithm can be improved by varying the gradient updating function, by comparing the Stochastic Gradient Descend algorithm (SGD) with the ADAM algorithm implemented as written in this article[2].

Then we also check if increasing the number of iterations of the Contrastive Divergence affects the results and helps the convergence.

Furthermore, we implement the so-called Centering Trick [3] that consists of using an Energy with visible and hidden values shifted respectively by two offsets μ and λ. As explained in detail in [3], choosing $\mu = \langle v \rangle_{data}$ and $\lambda = \langle h \rangle_{data}$, we expect that our RBM is invariant to any shift of variables and spins' flip. To actually implement this trick, one could simply work with a centered gradient as it has been shown in [3]:

$$\nabla W = \langle (v_d - \mu)(h_d - \lambda) \rangle - \langle (v_m - \mu)(h_m - \lambda) \rangle \tag{9}$$

$$\nabla a = \langle v_d \rangle - \langle v_m \rangle - \nabla W \lambda \tag{10}$$

$$\nabla b = \langle h_d \rangle - \langle h_m \rangle - \nabla W^T \mu \quad . \tag{11}$$

We update the offsets using a moving average with sliding factors ξ_μ and ξ_λ equals to 0.01, in order to update them smoother since they can be biased due to averages over finite samples.

3. RESULTS

Before implementing any additions to the RBM we start by varying the number of hidden layers for both $\{0, 1\}$ and $\{-1, 1\}$ bit notation to find the optimal configuration moving forward. For this we are looking at the weights of the visible nodes, trying to fine the expected alternating pattern of the AC chain (FIG. 2).

In all settings we can see a pattern of 4 reds and 4 blues, starting on one end with 2. This depicts the alternating pattern for the 4-digit ACs correctly. Thus we can assume that the output is acceptable for all cases, so we declare them all valid RBM machines.

First, the $\{0, 1\}$ notation shows a stricter alternating pattern than $\{-1, 1\}$ because by introducing a negative spin the distribution of possible states is wider spread and naturally any pattern is not as visible as with $\{0, 1\}$. From now on we are using the $\{0, 1\}$ notation.

Second, for every number of hidden layers compiled there is at least one that visually shows the alternating pattern. Surprisingly, only one hidden layer has enough parameters to correctly mirror the distribution accurately.

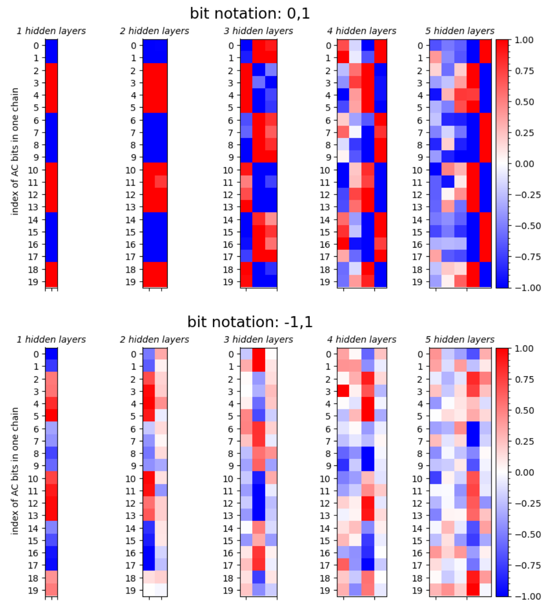

FIG. 1. Weights of visible layer plotted for 1 to 5 layers, for both $\{0, 1\}$ and $\{-1, 1\}$

3.1 SGD and ADAM

In this framework, we compare the Vannilla-SGD and ADAM algorithms with two different sizes of mini-batches: $m = 100, 500$. For Vanilla-SGD we use an adaptive learning rate starting from the value of 1. For ADAM we set the parameters according to the results of the respective Pytorch paper [2] with the difference that we modify the learning rate from 0.001 to 0.01 to improve convergence speed. With this setup, all the cases converge within 20 epochs. To observe the differences between different configurations, we compute the Delta between the average energy of the original data with the average energy of the data generated by the RBM of Eq.6. To improve the visualization of these Delta, we compute their means and standard deviations over the last ten epochs and put them in a residual plot. We also compute the LL calculated with the original data of Eq.7. As we can see in the energy plot of FIG. 2, the

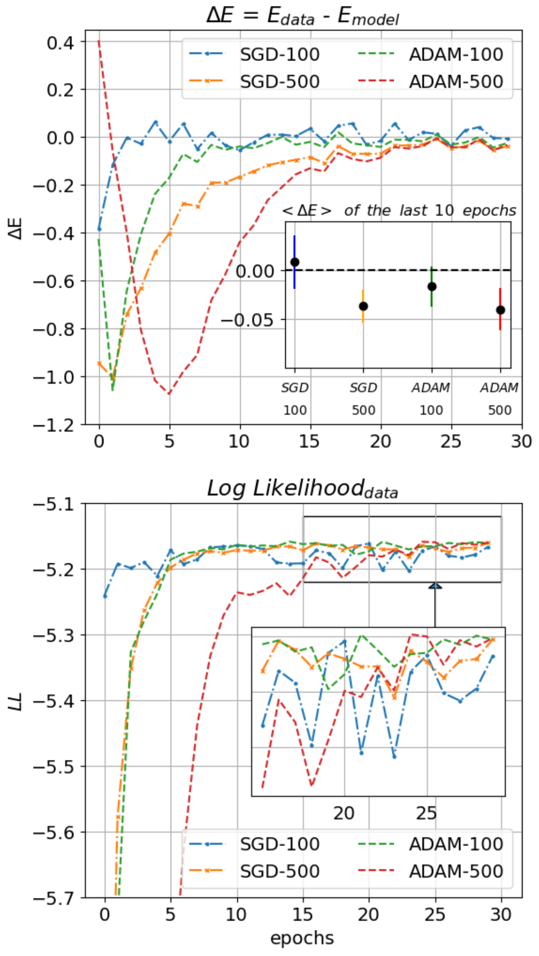

FIG. 2. ΔE, related residual plot and $LogLikelihood_{data}$ in the four cases studied: SGD with mini-batch $m = 100$, SGD with $m = 500$, ADAM with $m = 100$, ADAM with $m = 500$

RBM trained with Vanilla-SGD and a mini-batch of 100 samples converge very fast but is the less stable during the training. This is also confirmed by the large error bar in the residual plot. As we increase the mini-batch at 500 samples, the Vanilla-SGD delays the convergence, but then it is more stable as we expect. In ADAM's cases, we notice a similar behavior with respect to the Vanilla-SGD algorithm: the bigger the minibatch, the slower the convergence. Strangely enough, ADAM's ΔE with $m = 500$ reaches the value of -1 before the convergence and that makes it slower with respect to the others. We notice that the only two cases where the mean energy over the last ten epochs is compatible with 0 are the ones with a mini-batch size fixed to 100. In the residual plot, we notice that the ADAM with $m = 100$, ADAM-100, is the more stable algorithm since it has a smaller standard deviation. We plot the log-likelihood with respect to the original data to see whether the contribution of the log of the partition function Z helps the convergence of the training. As we can see, the LLs of Vanilla-SGD-500 and ADAM-100 are quite similar although the ΔE plots are different. From now on, if not specified, we will use ADAM-100 since is a valid trade-off between stability and convergence speed.

3.2 Contrastive Divergence

Now we evaluate the performance of the model when we change the number of Contrastive Divergence(CD) steps, which corresponds to perform extra iteration of Gibbs sampling.

We use 0, 1, 3, and 5 CD steps to see if the LL and the difference of mean energy between the data and the generated samples is affected for the different cases. In Fig. 3, we can see that the convergence of the model is reached before in the log-likelihood plot as we increase the number of contrastive divergence steps. On the other hand, our model seems to have the best results for both quality indicators when the number of CD steps is equal to 3. Indeed it is the model with an average energy over the last 15 epochs closer to 0 and the one that maximizes the LL function. For 5 CD steps, the model on the other hand seems appear to be getting worse.

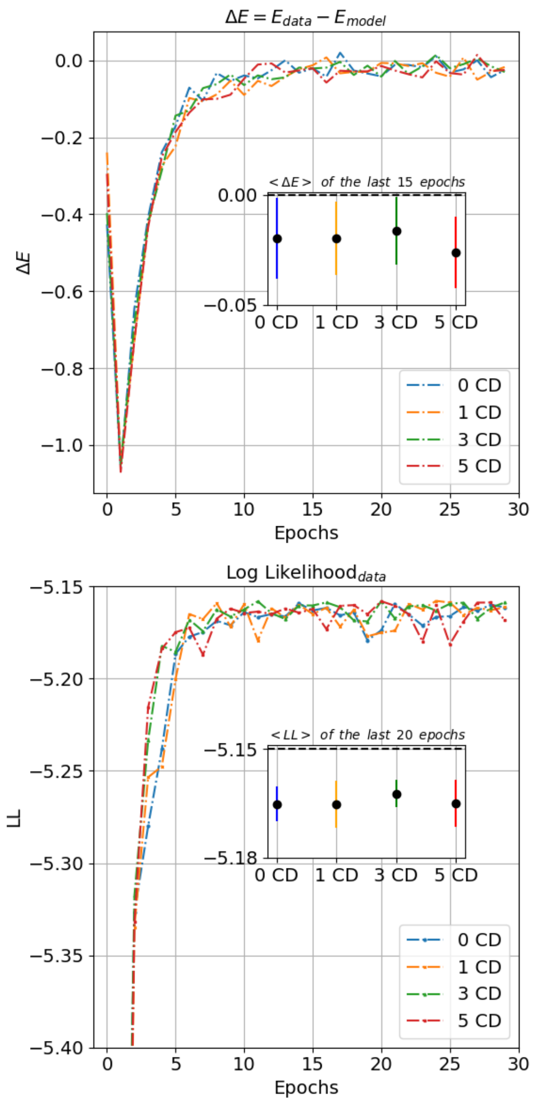

FIG. 3. ΔE and $LogLikelihood_{data}$ when we perform 0, 1, 3 and 5 Contrastive Divergence steps.

3.3 Centering Trick

Now we study how the Centering Trick affects the convergence of the optimization problem. In order to do that, we evaluate the performance of the model using different quality indicators: the difference between the energy of the data and the energy of the model, the log-likelihood, and also the error of the second momentum of Eq. 8.

Performing the Centering Trick improves the conditions of the optimization problem and reduces the variance of the gradient leading to faster and more stable solutions. We can see that from Fig. 4 where both the log-likelihood and the difference of energy have a faster and less noisy convergence with the Trick, even of only few epochs.

This is seen more easily in the case of the log-likelihood that in the case of the energies difference, which suggests that centering also causes a relevant change in the partition function Z. On the other hand, once the convergence is achieved ,the model parameters are both optimized very similarly. Nevertheless, we still observe a small improvement with respect to the non centered case. The centered log-likelihood is always bigger and less noisy than the one without Centering Trick applied, and that indicates an overall better performance.

As we can see in Fig. 5 the error with the Centered

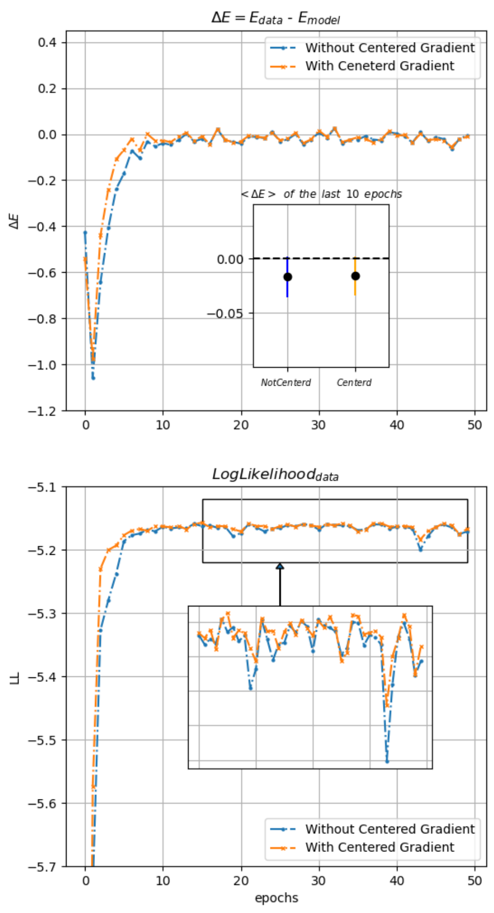

FIG. 4. ΔE and $LogLikelihood_{data}$ when we apply the Centering Trick and when we do not.

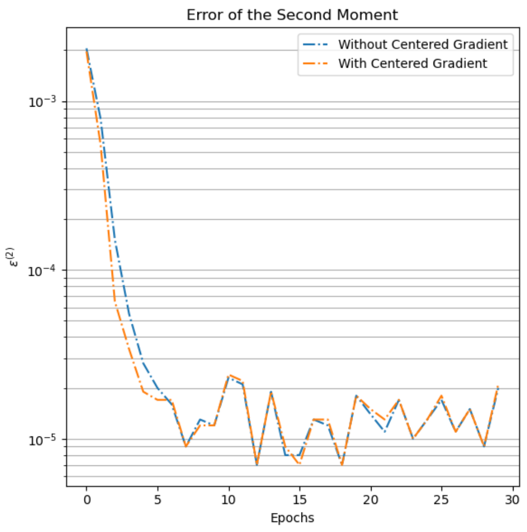

FIG. 5. Error of the second moment $\varepsilon^{(2)}$ with centering trick and without it.

Gradient achieves convergence earlier due to having a smaller variance of the gradient. But this doesn't mean the solution once we achieve convergence is better. As we can see both centered and not centered $\varepsilon^{(2)}$ are extremely similar once the optimal parameters have been found. Therefore, the Centering Trick, when considering the error of the second moment as a quality indicator, achieves a faster convergence but has no impact on improving the performance.

4. CONCLUSIONS

In this work, we have implemented a set of modern techniques for Restricted Boltzmann Machines applied to a simple example of linear proteins consisting of polar and non-polar amino acid chains.

First, we have shown that one hidden layer is enough to withhold the information on the sequence of polar and non-polar patterns when working in the one-hot encoding condition. Then we found a better representation of the RBM that permits us to instantly visualize patterns, by using spins in $\{0, 1\}$ instead of in $\{-1, 1\}$. Indeed a pattern in the weights and bias depict a more clear pattern with $\{0, 1\}$ bit representation.

We have seen that while implementing ADAM gradient ascend algorithm we are improving stability slightly, the cost for this benefit though is that the converging is lower.

On the other hand, two different techniques have been implemented in order to improve the convergence/performance of our RBM. The first one being the application of contrastive divergence steps shows both a better convergence and performance as the number of k steps is increased. We only have a sudden drop at 5 steps that shouldn't occur in theory as an increase in the number of k steps produces a reduction of the induced bias of the contrastive divergence. The other technique is centering, which has made our model converge faster in all quality indicators and maximized the log-likelihood. Therefore, both models have proven to be successful in getting better performance and faster convergence with respect to the not-centered model with no CD step iterations.

Finally, as far as the methods to evaluate performance are concerned, we have found that the average Energy is the most computationally efficient indicator with respect to the log-likelihood since it does not require to calculate Partition function, e.g. in the case of several CD the computation of Z becomes very demanding. The second error momentum is more precise than the differences of the mean energy because it takes into account explicitly the covariances of original and RBM data.

[1] A. Decelle, C. Furtlehner, and B. Seoane, "Equilibrium and non-equilibrium regimes in the learning of restricted boltzmann machines," *CoRR*, vol. abs/2105.13889, 2021.

[2] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[3] G. Montavon and K.-R. Müller, *Deep Boltzmann Machines and the Centering Trick*, pp. 621–637. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012.