Group 03 – Implementation of a Restricted Boltzmann Machine for protein-encoding

Veronica Bedin, Roben Bhatti, Diego Bonato, and Michail Sapkas
(Dated: April 2, 2023)

Generative models have become an essential tool in the past decade for evaluating a wide range of tasks, from corporate applications and medical image processing to large language models such as chatGPT. A specific type of generative model called Restricted Boltzmann Machine (RBM) has proven to be a very powerful tool in physics too. In this paper, we train a RBM on a dataset of artificially generated proteins with errors in the polarity pattern of the amino acid chain. Our goal is to explore the capability of the model in de-noising the data, that is to effectively capture the underlying rules of correct amino acid polarization. Different representations of data, various optimizers and Contrastive Divergence (CD) steps are tested. Finally, our results demonstrate the capability of the model to produce simple and interpretable results by visually inspecting the trained weights.

INTRODUCTION

Recently, machine learning algorithms have become widely used to tackle many problems, such as network anomaly detection in cybersecurity (Fiore, 2013 [6]), audio processing (Hardik et al. 2017 [7]) and high-dimensional data motion modelling in computer vision (Nie et al. 2015 [8]). For this reason, recent years have seen a significant increase in the use of Restricted Boltzmann Machines (RBMs) (Smolensky, 1986 [1]), which are generative models that rely on latent variables to model an input distribution.

We implement a RBM to spot and resolve the errors present in the dataset, generating correct amino acid chains, probing the interpretative capabilities of the model. We search for the best gradient descent optimizer and the impact of increasing the number of steps in the Contrastive Divergence algorithm, by training the model several times with different random seeds. Finally, by chosing the best hyperparameters, we attempt to explain the weight pattern of the model.

METHODS

The Dataset

The dataset consists of $10,000$ pseudo-proteins, each represented by 5 block of 4 bits each, with a total of 20 bits per protein. Each block encodes an amino acid and there are two equivalent ways of bit representation. If the bits belong in the set $\{+1, -1\}$, we refer to the *spins* representation, whereas another possibility is the set $\{0, +1\}$.

We represent the polarity of the amino acids using one-hot encoding. In our case, the states $(1, 0, 0, 0)$ and $(0, 1, 0, 0)$ refer to a polar amino acid, whereas the states $(0, 0, 1, 0)$ and $(0, 0, 0, 1)$ to a non-polar one.

The data were generated with this pattern: a polar amino acid must be followed by a non-polar one and vice versa. Errors in this pattern have also been introduced and account for approximately 22.6% of our dataset.

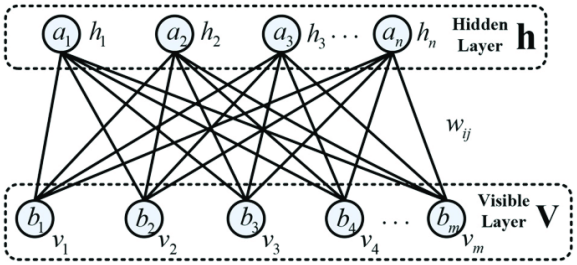

FIG. 1: The basic structure of a RBM, at the top the hidden layer, at the bottom the visible one, each one with its own units. Connecting them the weights w_{ij} (Chu et al. 2018 [2])

Theoretical Background

A RBM consists in a set of visible $\{v_i\}$ (i.e. the proteins) and hidden units $\{h_i\}$, as depicted in figure 1. The technical term for units, visible or hidden, that are allowed only binary values is *Bernoulli* units. The word *restricted* means that a set of weights $\{w_{i,j}\}$ encodes correlations between visible and hidden units only, with no interconnected units in the same layer. The goal of the RBM is to learn the underlying probability distribution of the data $P(v, h)$, that can be computed as:

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

where Z is the *partition function*, and $E(v, h)$ is the *energy* of the model given the visible and hidden units (v, h). We can compute the energy, adapting the definition from the Hopfield model, as in Mehta et al., 2019 [3]

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

where a_i and b_i are the the visible and hidden neurons *biases* respectively.

We initialize the parameters $\{a_i, b_i, w_{i,j}\}$ using the prescription by Glorot and Bengio, 2010[9], that is we randomly sample from a normal distribution with sigma dependent of the number of visible and hidden layers.

The model is trained with the Maximum Likelihood Estimation (MLE) method. This is done by maximizing the negative log likelihood cost function:

$$\mathcal{L}(\{\theta_i\}) = -\langle E((v,h);\{\theta_i\})\rangle_{data} - \log Z(\{\theta_i\}) \tag{3}$$

where $\{\theta_i\}$ are the parameters of the model.

The calculation of the partition function Z is in general an intractable task so we will maximize the equation 3 by computing the gradients with respect of the parameters $\{\theta_i\}$ and will attempt to treat Z as a marginalized probability distribution. One can show that these gradients take the following form (see Fischer et al. 2010 [4]):

$$\frac{\partial}{\partial\theta}\mathcal{L}(\theta|v_l) = \\ = -\sum_h p(h|v_l)\frac{\partial E(v_l,h)}{\partial\theta} + \sum_v p(v)\sum_h p(h|v)\frac{\partial E(v,h)}{\partial\theta} \tag{4}$$

In order to sample the second addend term of equation 4, we use k-steps of Contrastive Divergence (CD-k).

CD is an approximated version of Gibbs sampling, where the algorithm stops after k-steps instead of a long Markov Chain Monte Carlo (MCMC) sampling. Starting from an instance v_0 of the training set, we sample a hidden layer h from the probability distribution $p(h|v_0)$. Then we continue the sampling again by generating a fantasy layer v_t from $p(v_t|h)$ and so on.

After k-steps we can rewrite equation 4 as:

$$CD_k = -\sum_h p(h|v_0)\frac{\partial E(v_0,h)}{\partial\theta} + \sum_h p(h|v_k)\frac{\partial E(v_k,h)}{\partial\theta} \tag{5}$$

The terms above are expectation values and thus each parameter gradient can also be written, using equation 2, as:

$$\frac{\partial\mathcal{L}(\{W_{i\mu},a_i,b_\mu\})}{\partial W_{i\mu}} = \langle v_i h_\mu\rangle_{data} - \langle v_i h_\mu\rangle_{model}$$

$$\frac{\partial\mathcal{L}(\{W_{i\mu},a_i,b_\mu\})}{\partial a_i} = \langle v_i\rangle_{data} - \langle v_i\rangle_{model}$$

$$\frac{\partial\mathcal{L}(\{W_{i\mu},a_i,b_\mu\})}{\partial b_\mu} = \langle h_\mu\rangle_{data} - \langle h_\mu\rangle_{model} \tag{6}$$

where the subscript *data* refers to the values computed from the data, whilst *model* refers to the generated fantasy layers.

Model Evaluation Methods

The RBM code was implemented into a Python class, effectively allowing us to apply custom methods and have easy access and control to trainable variables and metrics. One custom method employed was storing the parameters that minimized the energy of the model at each epoch. As training metrics, we compute the log likelihood, the energy of the model and the data, and Shannon's entropy. Moreover, we define an accuracy measure by comparing our a priori knowledge of correct polarities patterns with the model's attempt to generate de-noised data by heuristic Energy amplification.

Different randomization should affect many steps of the training, such as the random initialization of the parameters $\{\theta_i\}$, the behaviour of the optimizers due to the shuffling of the data at the end of each epoch, and CD sampling due to probability acceptance in the activation function.

To evaluate the following optimizers: Stochastic Gradient Descent (SGD), Adam, Adagrad and RMSprop, we compute the log likelihood and accuracy values for each optimizer, using one hundred different random seeds. In the end we compute the mean and the standard deviation. For each optimizer the set of hyperparameters had to be different to ensure conversion.

In order to assess the role of randomization in CD, we follow a similar procedure as the one described before for up to 8 CD steps, using 9 random seeds for each step.

To explicitly compute the log likelihood (equation 3), we need to exactly compute the partition function Z. As mentioned before, this is usually intractable, but in our rather simple case this can be done by preserving the one hot encoding and thus computing $4^5 * 2^{hidden}$ Boltzmann terms. Finally, we chose the spins representation to run all tests.

RESULTS

Concerning the hidden units of the model, we heuristically found that three hidden neurons was the best choice for learning.

Our initial focus was to determine the best optimizer. All training was done using 20 epochs and it's worth noting that different data representations required different hyperparameter tuning in order to converge. The ϵ parameter for Adam and RMSprop needed to be larger than one using spins and very small using the other. Learning rates also needed to be adjusted, taking the value 1 for SGD, less than 0.01 for Adam and RMSprop and 0.1 for

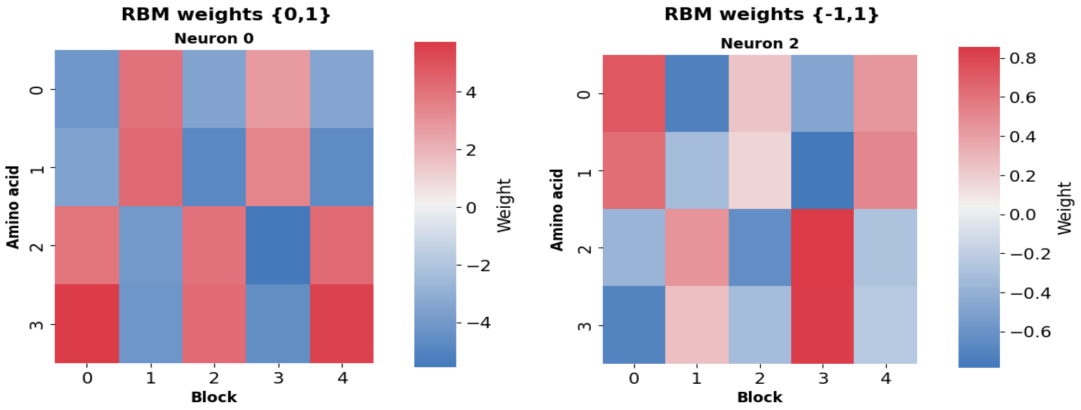

FIG. 2: Weights learned by the RBM using Adam as optimizer. By the alternation of positive and negative weights, we can see that the RBM has learned the pattern in both cases. The other neurons do not show any interpretable pattern.

Adagrad. Adam also converged faster than every other optimizer in just a few epochs.

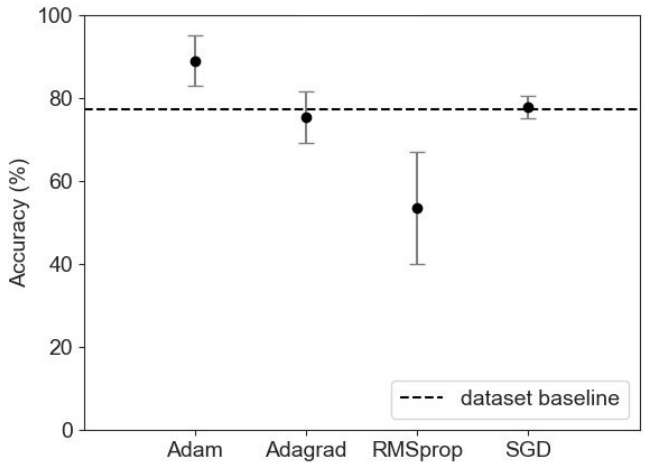

FIG. 3: Accuracy of the different optimizers: for each optimizer we trained 100 times the RBM with each time a different random seeds. For comparison, *dataset baseline* is shown, which is the accuracy of the initial dataset: 77.6% .

For Adam and RMSprop in particular, we notice that despite having the same mean log likelihood, their performances are completely opposite. Indeed, while Adam scores the best accuracy of $89.0 \pm 6.1\%$, RMSprop scores the worst, reaching only $53.5 \pm 13.6\%$. This suggests that, in this case, the log likelihood alone is not a good estimator of the performance of the model. Adagrad and SGD give similar mean accuracy results, even though with the same amount of errors as the original dataset, referred to as *dataset baseline* in the figures. RMSprop performs even worse, generating data with a lower accuracy than the original dataset: the instances generated contained more errors than the training set.

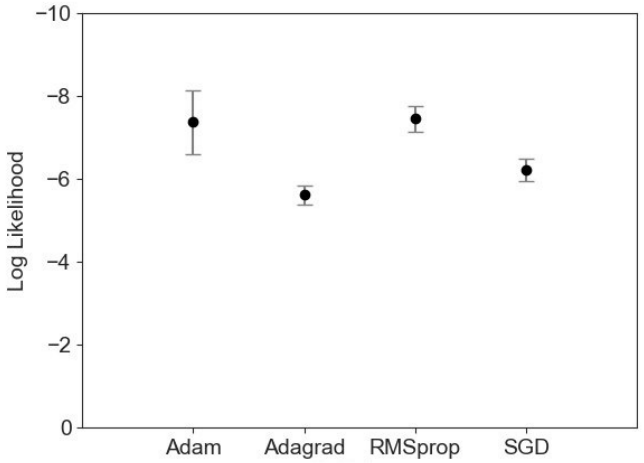

FIG. 4: Log likelihood of the different optimizers: for each optimizer we train the RBM with 100 different random seeds and compute mean and standard deviation.

The results, summarized in I, demonstrate that Adam outperforms the other optimizers evaluated in this study, with an accuracy of $Accuracy_{Adam} = 89.0 \pm 6.1\%$.

Optimizer	Accuracy[%]
Adam	89.0 ± 6.1
SGD	77.8 ± 2.6
Adagrad	75.5 ± 6.2
RMSprop	53.5 ± 13.6

TABLE I: Results of the RBM training. Adam is the best optimizer, RMSprop is the worst. The accuracy of the dataset is 77.4%

Therefore, Adam will be selected as the preferred optimizer for the rest of the analysis.

An interesting insight on what the RBM has learned is given by the heat maps shown in figure 2. Although we used the spins representation for the results of this paper, we also experimented with the 0,1 representation. There we can see two matrices plotting the weights of the RBM, trained with Adam, as a function of the five *blocks* that constitute the visible layers, and of the *amino acids*, which are the four one-hot encoded possible states. The left figure refers to the *hidden neuron 0* of the RBM trained with the $\{0, 1\}$ representation, while the right figure refers to the *hidden neuron 2* of the RBM trained with the spins representation. Both the heat maps are clearly showing the polar - non polar pattern of the data that has been learned by the machine and they only appear reversed because of the different learnt biases of the hidden and visible layers.

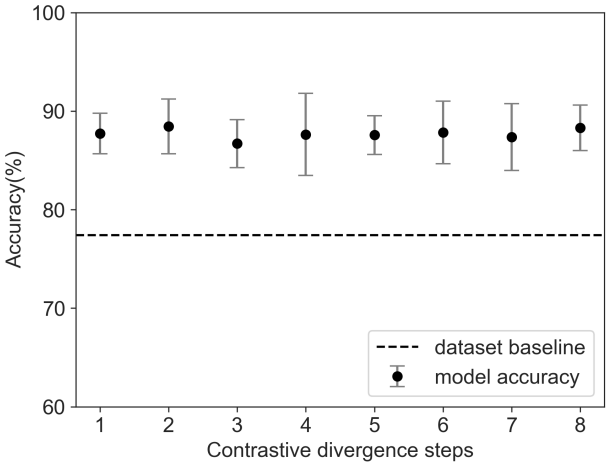

FIG. 5: Accuracy of Adam for different values of CD-steps. After choosing the best optimizer (Adam), we train the RBM with 9 different random seeds with different numbers of steps. We then compute the mean and standard deviation. For comparison, *dataset baseline* is shown, which is the accuracy of the initial dataset: 77.6% .

Investigating the performance of the Adam optimizer with respect to the number of CD-steps, our results indicate that increasing the number of steps does not have a significant impact on the accuracy metrics for CD, as depicted in 5. This confirms a result already discussed in literature (see Mehta et al., 2019 [3] or Hinton, 2010 [5]).

CONCLUSIONS

In our study we wanted to probe the capabilities of an RBM in predicting new data from a dataset that voluntarily contained errors. In order to do so, we searched for the best parameters to train the RBM. The best model is found with Adam as optimizer, reaching an accuracy of $89.0 \pm 6.1\%$ and three hidden layers. Using more than one steps of CD does not affect accuracy, as already mentioned in literature (Hinton, 2010 [5]).

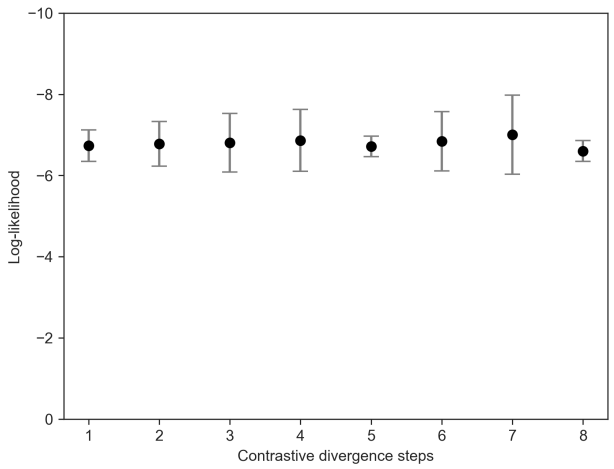

FIG. 6: Log likelihood for different steps of CD using Adam. After choosing the best optimizer (Adam), we train the RBM with 9 different random seeds with different numbers of steps. We then compute the mean and standard devation.

In future works, we could include experimentation with amplification or attenuation of the energy during CD steps, which in the physics context can be interpreted as temperature.

The RBM model demonstrated its high interpretative power, even though it's important to seek through the wide range of possible (hyper) parameters available to get the best performance.

[1] Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. In D. Rumelhart and J. McClel- land (Eds.), Parallel distributed processing, vol. 1, chapter 6, 194–281. Cambridge: MIT Press (1986).

[2] Chu, Yaqi Zhao, Xingang Zou, Yijun Xu, Peter Han, Jianda Zhao, Yiwen. (2018). A Decoding Scheme for Incomplete Motor Imagery EEG With Deep Belief Network. Frontiers in Neuroscience. 12. 680. 10.3389/fnins. 2018. 00680.

[3] 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,

[4] Asja Fischer and Christian Igel, Empirical Analysis of the Divergence of Gibbs Sampling Based Learning Algorithms for Restricted Boltzmann Machines, 2010

[5] Geoffrey Hinton, A Practical Guide to Training Restricted Boltzmann Machines, 2010

[6] Ugo Fiore, Network anomaly detection with the restricted Boltzmann machine, 2013

[7] Hardik B. Sailor, Dharmesh M. Agrawal, and Hemant A.

Patil, Unsupervised Filterbank Learning Using Convolutional Restricted Boltzmann Machine for Environmental Sound Classification, 2017

[8] Siqi Nie et al., A generative restricted Boltzmann machine based method for high-dimensional motion data modeling, 2015

[9] Glorot, Xavier, Bengio, Yoshua, 2010. Understanding the difficulty of training deep feedforward neural networks. In: Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, pp. 249–256.