2308 - Optimal recipe for analyzing one-hot encoded data with Restricted Boltzmann Machines: parameter selection and evaluation of performance with adversarial accuracy and log-likelihood

Brisigotti Erica, Chueva Ekaterina, Pacheco Garcia Sofia, and Sahputra Nadillia
(Dated: April 2, 2023)

There is no free lunch, but at least there are free recipes. In this work, we share our optimal recipe for analyzing one-hot encoded data. It uses Restricted Boltzmann Machines and might be helpful to the reader for various applications. We not only show the best parameters, but also justify most of the choices quantitatively. Furthermore, the choice of the gradient descent algorithm is based on two different estimators. Our results agree with Occam's razor: the optimal recipe is the simplest one and consists of RMSprop, binary encoding and no centering trick.

INTRODUCTION

This work focuses on solving a generative task that requires an unsupervised machine learning technique. The ultimate goal is to train a neural network that is interpretable and able to generate new denoised data. For these reasons, we use Restricted Boltzmann Machines (RBM). The original dataset used to train RBM is synthetic, yet it resembles data in biological physics applications. The choices of parameters and methods are motivated for reproducibility. We choose the optimal gradient descent algorithm (ADAM, RMSprop or Vanilla) and parameters of RBM. The choices are made based on theory and experimental results that we obtained. For estimating the performance, we use the adversarial accuracy indicator (accuracy MSE) and maximum likelihood estimation (log-likelihood). Such estimators are used to both set the parameters and find the best-performing gradient descent algorithm.

Data

The analysed dataset has dimensions $(10000, 20)$. Each line can be divided into $G = 5$ blocks, each one belonging to a set $\{v_k\}_{k=1}^{N_S}$ of $N_S = 4$ possible one-hot encoded states (each one of length $A = 4$):

$$\{v_k\}_{k=1}^{N_S} = \{[1,0,0,0]\ [0,1,0,0]\ [0,0,1,0]\ [0,0,0,1]\} \quad (1)$$

Noise is later introduced with a 10% probability of changing from the correct state to another.

This dataset could at first glance seem artificial, but its structure is very similar to the data that is obtained by translating (for machine learning purposes) categorical data (such as amino-acids) into binary variables with one-hot encoding. Similarly, bipolar encoding can be chosen, instead of the original binary encoding, to potentially increase the ML algorithm efficiency.

Restricted Boltzmann Machines

A Restricted Boltzmann Machine is a neural network that can be represented as bipartite graph of two non-interacting layers of variables: the hidden variables $\mathbf{h}$ and visible variables $\mathbf{v}$, both consisting of discrete binary stochastic units (that are referred to as Bernoulli units [1]). The layers have weighted connections, which are encoded in the visible bias $\mathbf{a}$, in the hidden bias $\mathbf{b}$ and in the weight matrix $\mathbf{w}$. The energy of a configuration of visible and hidden unit $(\mathbf{v}, \mathbf{h})$ is defined as:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{N_v} a_i v_i - \sum_{j=1}^{N_h} b_j h_j - \sum_{i=1}^{N_v}\sum_{j=1}^{N_h} h_j v_i w_{ij}, \quad (2)$$

where a_i, v_i, b_j, h_j and w_{ij} are the components of the aforementioned $\mathbf{a}$, $\mathbf{v}$, $\mathbf{b}$, $\mathbf{h}$ and $\mathbf{w}$; N_v and N_h are the number of visible and hidden units, respectively [2].

The probability of a particular configuration $(\mathbf{v}, \mathbf{h})$ can be computed from the energy:

$$p(\mathbf{v}, \mathbf{h}) = \frac{1}{Z} e^{-E(\mathbf{v},\mathbf{h})} \quad (3)$$

where the normalization factor $Z = \sum_{\mathbf{v},\mathbf{h}} e^{-E(\mathbf{v},\mathbf{h})}$ is the partition function of the system. The probability $p(\mathbf{v}, \mathbf{h})$ can be marginalize to calculate the probability for a particular configuration of the visible variables: $p(\mathbf{v}) = \sum_{\mathbf{h}} p(\mathbf{v}, \mathbf{h})$.

Contrastive Divergence

A contrastive divergence (CD-k) algorithm is used to train the RBM. This algorithm consists of two steps: the positive phase, when $\mathbf{h}$ is generated from $\mathbf{v}$, and negative phase, when $\mathbf{v}$ is generated based on $\mathbf{h}$. At each step, the CD algorithm generates new variables randomly based on the knowledge that is been acquired up until that point. The choice of a number of contrastive divergence steps $k \geq 1$ is non-trivial since it is possible to obtain good results for RBM for just $k = 1$ [2]. The generation of $\mathbf{h}$ is based on the probability given by the following formula:

$$p(h_j|\mathbf{v}) = \sigma\left(-\delta_{GAP}(b_j + \textstyle\sum_{i=1}^{N_v} w_{ij} v_i)\right), \quad (4)$$

where σ is the sigmoid function and δ_{GAP} is variable dependent on the choice of encoding for the Bernoulli units:

$$\delta_{GAP} = \begin{cases} 1 & \text{for original encoding} \\ 2 & \text{for bipolar encoding} \end{cases} \quad (5)$$

The RBM is run, in the early stages, for both cases in order to establish the more easy-to-interpret encoding of the Bernoulli units.

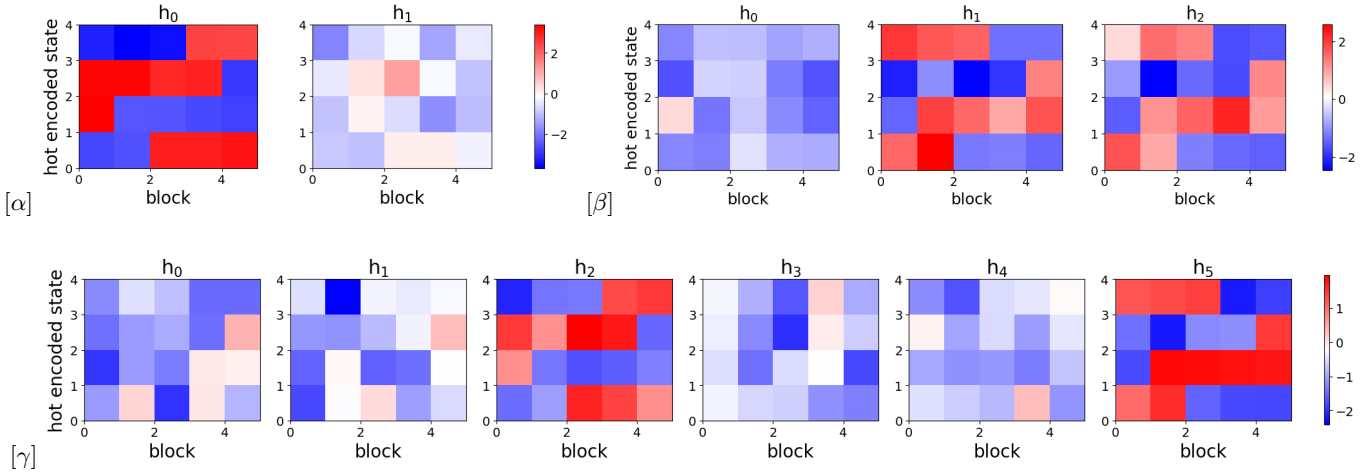

FIG. 1: Graphical representation of the weights of each unit of hidden variable. The different plots correspond to the results of RBM with different numbers of hidden variables: $N_h = 2$ [α], $N_h = 3$ [β], $N_h = 6$ [γ]. These examples are obtained by setting 100 epochs, with original encoding, without centering trick, for k=5.

The generation of $\mathbf{v}$ can be implemented in two different ways. If the one-hot encoding is ignored, it is similar to the generation $\mathbf{h}$ described above. Instead, we opt for using and preserving the one-hot encoding by generating every chunk of $\mathbf{v}$ as one of the one-hot encoded states $\{v_k\}_{k=1}^{N_S}$. The probability of having the one-hot encoded state v_k in chunk c is computed as:

$$p_c^k = \frac{B_c^k}{\sum_{l=1}^{N_S} B_c^l} \text{ where } B_c^k = \exp(-\delta_{GAP} \cdot E_c^k(\mathbf{h})) \quad (6)$$

The energy term $E_c^k(\mathbf{h})$ due to having the one-hot encoded vector v_k in chunck c was calculated as:

$$E_c^k(\mathbf{h}) = -\left(\mathbf{a}_c + \mathbf{w}_c \mathbf{h}\right) v_k \quad (7)$$

where $\mathbf{a}_c$ and $\mathbf{w}_c$ are, respectively, the sub-vector of $\mathbf{a}$ and sub-matrix of $\mathbf{w}$ corresponding to chunk c.

METHODS

This section aims to describe the methods implemented in the analysis. In addition to that, we will justify the choices of parameters that were made in the analysis.

Mini-Batch Size

RBM algorithms prefer smaller batch sizes, between 10 and 100, especially when applying gradient descent algorithms [2]. Therefore, the mini-batches size is set to 50, for a total of 200 mini-batches per epoch since the number of lines in the original dataset is $N_L = 10000$.

Number of Hidden Units

The choice of number of hidden units N_h is based on the fact that the weights can take on both positive and negative values. Therefore, a RBM with N_h hidden variables can describe at least 2^{N_h} possible behaviors. Therefore, we estimate the number of hidden variables $N_h = \log_2(N_S) = 2$ based on the number of one-hot encoded states $N_S = 4$. The choice of $N_h = 2$ is confirmed by the visual comparison in figure (1), which shows redundancy between weight tables corresponding to different hidden variables for $N_h > 2$.

Adversarial Accuracy Indicator

The adversarial accuracy indicator is used to measure the accuracy of the RBM algorithm. It measures the minimum Euclidean distance between the original data and the generated result to indicate the their closeness. We first compute the mean distance of each set of samples to every other set of samples in same the original dataset $d_{SS}(m)$. We also calculate the mean distance of every set of samples from the original dataset to every set of samples in the fantasy dataset $d_{ST}(m)$. [3]

$$\mathcal{A}_S = \frac{1}{N_L} \sum_m^{N_L} \mathbf{1}\left(d_{SS}(m) < d_{ST}(m)\right) \quad (8)$$

where $N_L = 10000$ is the number of lines in the original dataset. Similarly, we compute the mean distance using each set of samples from fantasy dataset to another set of samples from the same dataset $d_{TT}(m)$ and measures the distance also to the original dataset $d_{TS}(m)$.

$$\mathcal{A}_T = \frac{1}{N_L} \sum_m^{N_L} \mathbf{1}\left(d_{TT}(m) < d_{TS}(m)\right) \quad (9)$$

To take into account both quantities, we calculate their Mean Squared Error (MSE) with the equation below:

$$\mathcal{E}^{\text{AA}} \equiv \left(\mathcal{A}_S - 0.5\right)^2 + \left(\mathcal{A}_T - 0.5\right)^2 \quad (10)$$

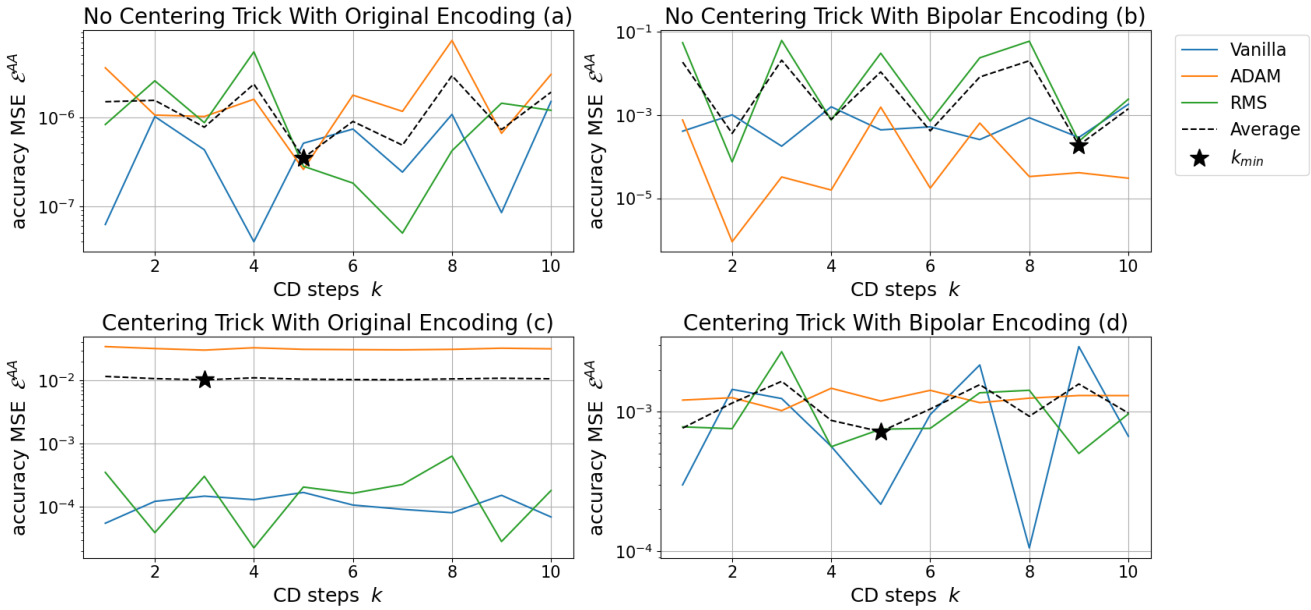

FIG. 2: Graphical representation of the accuracy MSE $\mathcal{E}^{AA}$ per number of CD steps k for all combinations of the other parameters. All graphs are obtained by setting 100 epochs. All 3 gradient descent algorithms are represented, together with their average and its minimum.

Maximum likelihood estimation

Maximum Likelihood Estimation is another useful tool that we implement to evaluate the performance of the algorithms for different combinations of parameters.

In the context of RBMs, the log-likelihood function quantifies the goodness of the fit between original and generated data. MLE selects the optimal parameters to be the ones corresponding to maximum likelihood (making generated data most like the original one).
The log likelihood is computed as:

$$\mathcal{L} = \lim_{\beta_k \uparrow 1} \frac{1}{N_L} \sum_{\mathbf{v},\mathbf{h}} \log\left(p_k(\boldsymbol{v},\boldsymbol{h})\right) \tag{11}$$

where:

$$p_k(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{Z_k} \exp\left(-\beta_k \cdot \delta_{GAP} \cdot E(\boldsymbol{v},\boldsymbol{h})\right) \tag{12}$$

and the values of coefficient $\{\beta_k\}$ are varied such that:

$$0 < \beta_0 \quad \wedge \quad \beta_k < \beta_{k+1}\ \forall k \quad \wedge \quad \lim_{k\uparrow+\infty} \beta_k = 1 \tag{13}$$

Centering Trick Implementation

We also applied the centering trick introduced in [3]. It is based on the idea of shifting visible and hidden variables in the RBM by two sets of offset parameters ($\boldsymbol{\mu}$= $[\mu_1, ..., \mu_{N_v}]$ and $\boldsymbol{\lambda}$= $[\lambda_1, ..., \lambda_{N_h}]$). We choose for $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ to be the expectation values of the visible and hidden units, obtaining the Montavon and Müller model. Therefore, the centering trick parameters may depend on the model parameters and, thus, change during training. For this reason, the centering trick parameters must be updated for every mini-batch. [4].

Choice of other parameters

Lastly, we exploit the properties of accuracy MSE $\mathcal{E}^{AA}$ and log-likelihood $\mathcal{L}$ to choose the remaining parameters. For each estimator, we identify the optimal combination of number of CD steps k, choice of units (original or bipolar) and centering trick (if to be applied or not) by analyzing the average performance ($\overline{\mathcal{E}^{AA}}$ or $\overline{\mathcal{L}}$) of the three gradient descent algorithms.
The average performance is further summarized by considering just one of its extrema: the minimum of the accuracy MSE or the maximum of the log-likelihood. This process allows to identify an optimal value of CD steps k (k_{min} or k_{max}) for that particular configuration.

Finally, we choose the best configuration of parameters per estimator based on the following considerations:

- based on accuracy MSE, the optimal configuration is the one identified by the smallest value of $\min(\overline{\mathcal{E}^{AA}})$. Case [a], with original encoding and no centering trick, is the clear winner based on figure (2). Therefore, for the analysis in terms of accuracy MSE, we choose to proceed with original encoding, no centering trick and number of CD steps $k = 5$.

- in terms of log-likelihood, the optimal configuration is the one identified by the largest value of $\max(\overline{\mathcal{L}})$. Case [a], with original encoding and no centering trick, is the optimal one, as shown in figure (3) For the analysis in terms of log-likelihood, we choose to proceed with original encoding, no centering trick and number of CD steps $k = 7$.

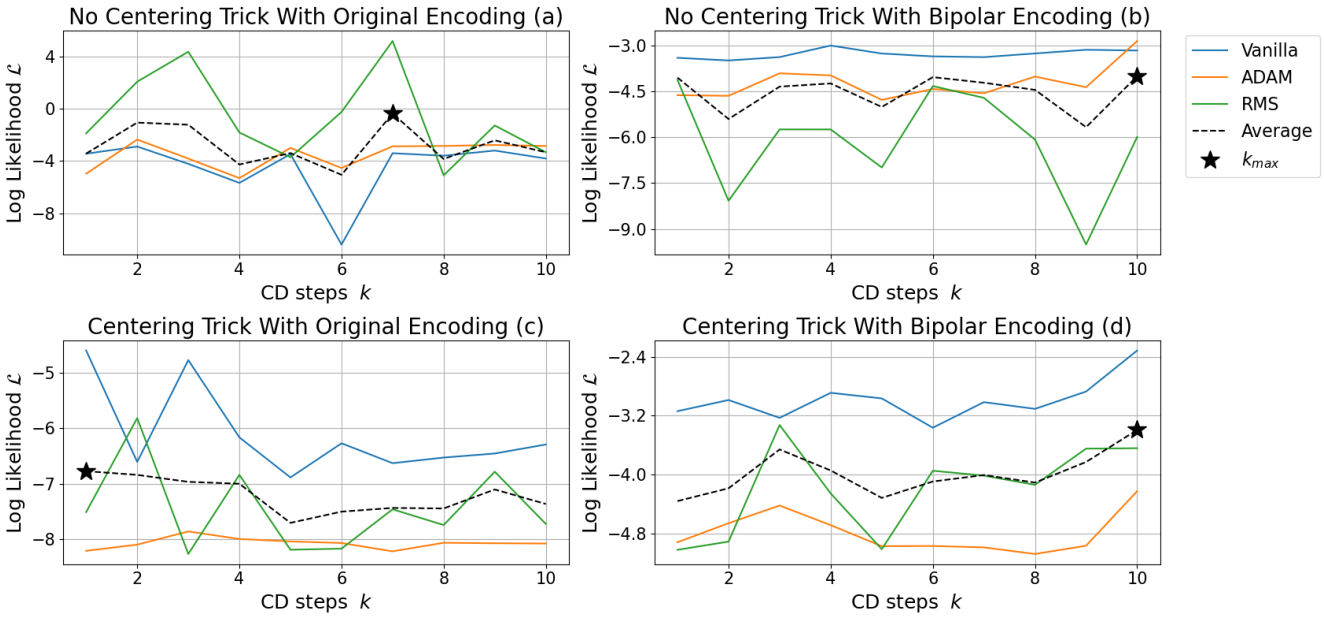

FIG. 3: Graphical representation of the log-likelihood $\mathcal{L}$ per number of CD steps k for all combinations of the other parameters. All graphs are obtained by setting 100 epocs. All three gradient descent algorithms are represented, together with their average and its maximum.

RESULTS

Having set the majority of the parameters, we now analyze the behaviour of 3 of the main gradient descent algorithms with respect to time (number of epochs): ADAM, RMSprop and Vanilla gradient descent. Their performance is evaluated quantitatively by computing both the accuracy MSE and the log-likelihood per epoch for an extended amount of time:

- in terms of accuracy MSE, the optimal algorithm is RMSprop, as shown in figure (4). Its optimal performance is reached around 200 ± 10 epochs. We also notice that, after 20 ± 10 epochs, the accuracy trend plateaus.
- based on log-likelihood, the best performing algorithm is RMSprop, as shown in figure (4). Its optimal performance is reached around 20 ± 10 epochs.

For both cases, weights and generated data at optimal time are displayed, respectively, in figures (5) and (6). In both cases, the RMSprop algorithm returns the best results. The only difference between the two methods is the otpimal number of epochs.

CONCLUSIONS

The results lead the following conclusions:

- in figure (5) we see that the optimal RBM for accuracy MSE performs better than the one for log-likelihood. This follows from the one of the two parameter that differ between the two RBMs, the number of epochs, which is 200 ± 10 for the accuracy RBM and instead is 20 ± 10 for the log-likelihood. In both cases, the majority of the generated data reflects the polar/nonpolar correlation that was coded into it at generation, before adding noise in 10%. The generation is successful.
- from figure (6)we notice that, for a small number of epochs (like 20 ± 10, in the log-likelihood case), weights corresponding to different hidden units are similar to each other. They evolve with time to be more distinguishable (like for 200 ± 10 epochs in the accuracy case).

To further summarize, we give out our unique recipe which is based on figure (4): RMSprop, original encoding, no centering trick, 7 CD steps and 20 epochs. We choose this value because it is the optimal value of the log-likelihood and also corresponds to the start of the plateau for the accuracy MSE.

REFERENCES

1. Mehta, P. *et al.* A high-bias, low-variance introduction to Machine Learning for physicists. *Physics Reports* **810,** 1–124 (May 2019).
2. Hinton, G. E. in *Neural Networks: Tricks of the Trade (2nd ed.)* 599–619 (Springer, 2012). ISBN: 978-3-642-35288-1.
3. Decelle, A., Furtlehner, C. & Seoane, B. Equilibrium and non-equilibrium regimes in the learning of restricted Boltzmann machines. *Journal of Statistical Mechanics: Theory and Experiment* **2022,** 114009 (Nov. 2022).
4. Bortoletto, M. & Baiesi, M. Study of performances for Restricted Boltzmann Machines. **2021,** 59 (Sept. 2021).

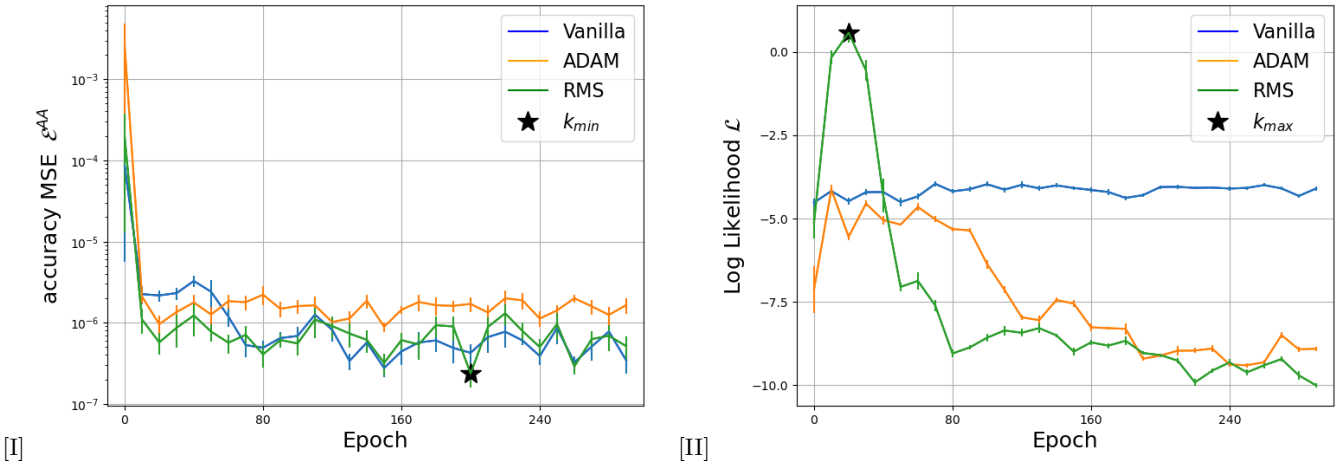

FIG. 4: Graphical representation of: accuracy MSE $\mathcal{E}^{AA}$ [I] and log-likelihood $\mathcal{L}$ [II] per epoch for the 3 gradient descent algorithms. The examples are obtained with original units and no centering trick. We set $k = 5$ for the accuracy MSE, and $k = 7$ for the log-likelihood, as established in the methods section. The results are averaged every 10 epochs to display a clearer plot.

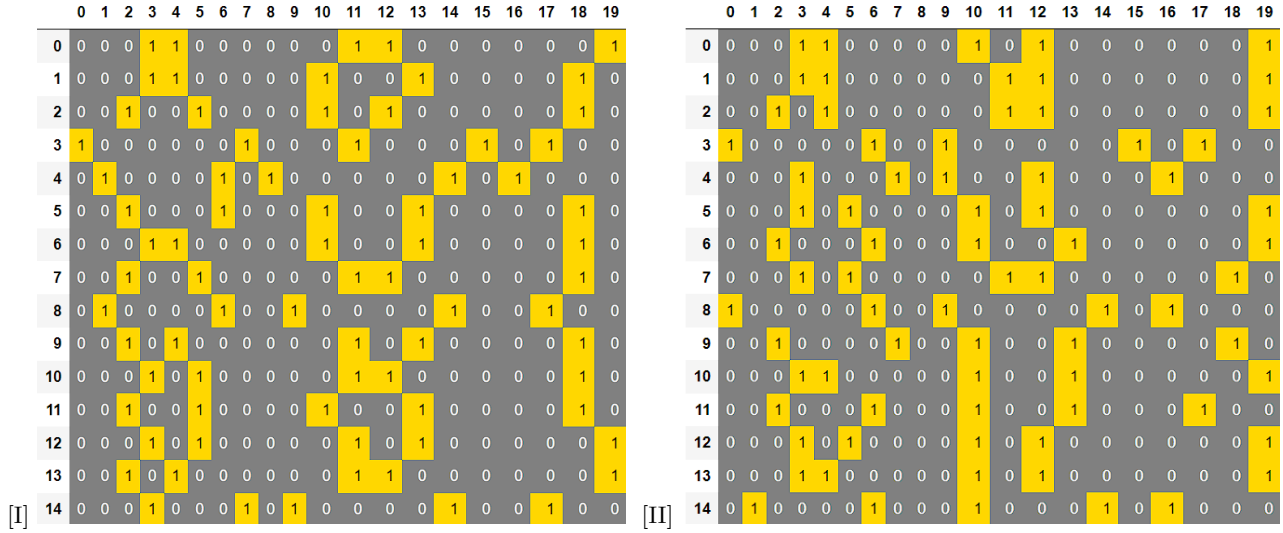

FIG. 5: Representation of the randomly generated data for the best performing algorithm, RMSprop. For the accuracy MSE $\mathcal{E}^{AA}$ plot [I], we set $k = 5$, with original encoding and no centering trick, after 200 epochs. For the log-likelihood $\mathcal{L}$ plot [II], the RBM is run after setting $k = 7$, with original encoding and no centering trick, for a total of 20 epochs.

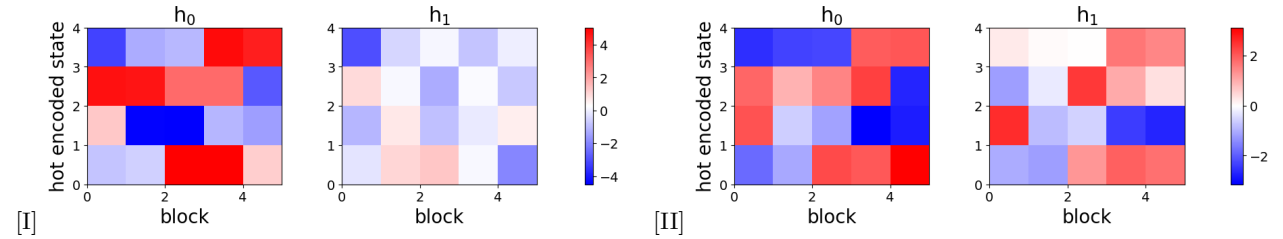

FIG. 6: Visualization of the weights per hidden variable for the best performing algorithm, RMSprop. For the accuracy MSE $\mathcal{E}^{AA}$ plot [I], we set $k = 5$, with original encoding and no centering trick, after 200 epochs. For the log-likelihood $\mathcal{L}$ plot [II], the RBM is run after setting $k = 7$, with original encoding and no centering trick, for a total of 20 epochs.