GROUP 08 - Restricted Boltzmann Machines

Brocco Luca, Cafagno Samuele, Cavallin Jonathan, and Di Lucia Sofia
(Dated: May 15, 2025)

In this paper we present the results obtained applying a Restricted Boltzmann Machine (RBM) to the MNIST dataset, specifically focusing on the task of generating digits 1, 4, and 7, which share notable similarities. We examine how the RBM's performance is influenced by different choices of key hyperparameters such as the number of hidden units, the number of Contrastive Divergence steps and the choice of the likelihood maximization algorithm with its regularization parameter. Additionally, we investigate alternative models to the usual Bernoulli case: the Potts model, where only one of the hidden neurons can be turned on, and the Spins model, where each neuron can be in state $\{\pm 1\}$. Moreover, we analyze the contribution of the different combinations of hidden states to the generation of the digits, which shows interesting correlations. Finally, we introduce a modification to the Contrastive Divergence algorithm that forces hidden states to transition between digits' representation. We show that this modification leads to an increase in energy for intermediate configurations, providing an explanation for the fact that such transitions are unlikely to be observed with standard Contrastive Divergence.

1. INTRODUCTION

In the context of unsupervised Machine Learning, Restricted Boltzmann Machines (RBMs) are a powerful type of stochastic neural networks capable of learning a probability distribution of a set of input data. The learnt model can generate new data that are statistically similar but distinct from the training set, thus functioning as a generative model.

RBMs are two-layer networks in which input neurons, also called visible units, are connected to the hidden layer units using symmetrically weighted connections. No direct interactions can occur between visible or hidden units, as illustrated in Fig. 1.

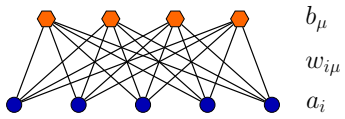

FIG. 1: RBM scheme with visible units x_i and hidden units z_μ connected through interactions of the form $w_{i\mu}x_i z_\mu$.

The network assigns a scalar energy value to each joint configuration of visible and hidden units

$$E(\mathbf{x},\mathbf{z}) = -\sum_{i=1}^{D} a_i x_i - \sum_{\mu=1}^{L} b_\mu z_\mu - \sum_{i=1}^{D}\sum_{\mu=1}^{L} x_i w_{i\mu} z_\mu \quad (1)$$

where x_i represents the visible units and z_μ the hidden units, which in the usual Bernoulli model can assume binary boolean values {0,1}, a_i and b_μ are bias weights and $w_{i\mu}$ are the interaction weights between visible and hidden units. In this work the number of visible units is fixed to $D = 784$, the number of bits of each digit of the MNIST dataset.

The joint probability distribution of observing state $(\mathbf{x},\mathbf{z})$ is defined in terms of this energy function as

$$p(\mathbf{x},\mathbf{z}) = \frac{e^{-E(\mathbf{x},\mathbf{z})}}{Z} \quad (2)$$

where

$$Z = \sum_{\{(\mathbf{x},\mathbf{z})\}} e^{-E(\mathbf{x},\mathbf{z})} \quad (3)$$

is the partition function of the system ensuring proper normalization. In particular, the marginal probability distribution of observing the visible state $\mathbf{x}$ is found summing over all 2^L possible hidden states

$$p(\mathbf{x}) = \frac{1}{Z}\sum_{\{\mathbf{z}\}} e^{-E(\mathbf{x},\mathbf{z})} \quad (4)$$

With this formulation, lower energy values correspond to more probable states and vice-versa. By properly tuning weights and biases, the energy of training images can be minimized, thereby raising their associated probability and making them more likely to be generated.

2. METHODS

2.1. RBM's training and Contrastive Divergence

A fundamental measure of the RBM performance is the likelihood $\mathcal{L}$, which quantifies how well the model, described by parameters θ, fits observed data. Usually the logarithm of the likelihood is used and we have, given a dataset of M samples

$$\ln \mathcal{L} = \ln \prod_{i=1}^{M} p(x_i|\theta) = \sum_{i=1}^{M} p(x_i|\theta) \quad (5)$$

RBMs are trained by maximizing the log-likelihood, or, equivalently, by minimizing the negative log-likelihood, using Stochastic Gradient Descent (SGD) or its variants. To do so, it is necessary to compute the gradients of the log-likelihood with respect to the learnable parameters. From Eq. 1, 4 and 5 it can be found [1]:

$$\begin{aligned}\frac{\partial \ln \mathcal{L}(\theta)}{\partial w_{i\mu}} &= \langle x_i z_\mu \rangle_{\text{data}} - \langle x_i z_\mu \rangle_{\text{model}} \\ \frac{\partial \ln \mathcal{L}(\theta)}{\partial a_i} &= \langle x_i \rangle_{\text{data}} - \langle x_i \rangle_{\text{model}} \\ \frac{\partial \ln \mathcal{L}(\theta)}{\partial b_\mu} &= \langle z_\mu \rangle_{\text{data}} - \langle z_\mu \rangle_{\text{model}}\end{aligned} \tag{6}$$

where $\theta = \{w_{i\mu},\ a_i,\ b_\mu\}$.

The gradients consist of a postive term that depends on the data and a negative term that depends on the model's predicted distribution. The first term is straightforward to compute using the dataset. The challenge especially lies in computing the second term of the first gradient, which represents the correlation between a visible node and a hidden node. It requires summing over an exponential number of configurations or using Markov Chain Monte Carlo (MCMC) methods that tipically require many steps to converge to stability.
To address this, Contrastive Divergence (CD) is used to approximate this gradient efficiently by running parallel MCMC simulations. From the dataset, a small minibatch of N points $\mathbf{x^0}$ is selected (corresponding to the minibatch used to estimate gradients with SGD), which serves as the starting point for a Gibbs chain that runs for a fixed number of steps N_t. Each iteration t of Gibbs sampling involves sampling $\mathbf{z^t}$ from the conditional distribution $p(\mathbf{z}|\mathbf{x^t})$, followed by sampling $\mathbf{x^{t+1}}$ from $p(\mathbf{x}|\mathbf{z^t})$ (see Fig. 2).

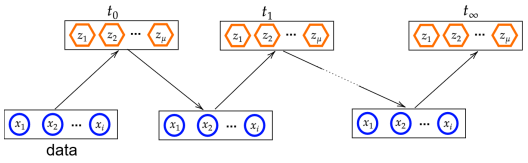

FIG. 2: Contrastive Divergence scheme: we alternate between updating the hidden and visible layers. Each iteration t corresponds to the number of passes between the visible and hidden states.

After N_t iterations, the gradient is estimated as

$$\frac{\partial \ln \mathcal{L}(\theta)}{\partial w_{i\mu}} = \langle x_i z_\mu \rangle_{\text{data}} - \frac{1}{N} \sum_{\mathbf{x^{N_t}} \in \text{minibatch}} x_i^{N_t} z_\mu^{N_t} \tag{7}$$

and similarly for the bias weights.

The RBM's bipartite structure simplifies the Gibbs sampling: since visible units are conditionally independent given the hidden units (and vice versa) we can update an entire layer in parallel based on its conditional distribution:

$$\begin{aligned}p(\mathbf{x}|\mathbf{z}) &= \prod_i p(x_i|\mathbf{z}) \\ p(\mathbf{z}|\mathbf{x}) &= \prod_\mu p(z_\mu|\mathbf{x})\end{aligned} \tag{8}$$

with

$$\begin{aligned}p(x_i = 1 \mid \mathbf{z}) &= \sigma(a_i + \sum_\mu w_{i\mu} z_\mu) \\ p(z_\mu = 1 \mid \mathbf{x}) &= \sigma(b_\mu + \sum_i w_{i\mu} x_i)\end{aligned} \tag{9}$$

where σ is the sigmoid activation function

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{10}$$

Contrastive Divergence can be interpreted as forming "basins of attraction" around the data: the idea is that starting from desired equilibrium configurations and making N_t steps, we explore nearby configurations representative of the dataset when the machine is learned; otherwise, if the chains flow away they will "teach" the RBM how to adjust the parameters.[2]

2.2. Regularization

To prevent overfitting and improving the generalization of the model, it is common to supplement the log-likelihood with an additional regularization term. Usually this works by adding a function that penalizes large weights. In this work we choose a L1 regularization (LASSO), thus the learning problem becomes

$$\arg\min_{w_{i\mu},\ a_i,\ b_\mu} g(\theta) \tag{11}$$

with

$$g(\theta) = -\ln \mathcal{L}(\theta) + \gamma \sum_{i,\ \mu} |w_{i\mu}| + |a_i| + |b_\mu| \tag{12}$$

This implies the presence of an additional term for the computation of the gradients in SGD:

$$\begin{aligned}\frac{\partial g(\theta)}{\partial w_{i\mu}} &= \frac{\partial \ln \mathcal{L}(\theta)}{\partial w_{i\mu}} + \gamma \cdot \text{sgn}(w_{i\mu}) \\ \frac{\partial g(\theta)}{\partial a_i} &= \frac{\partial \ln \mathcal{L}(\theta)}{\partial a_i} + \gamma \cdot \text{sgn}(a_i) \\ \frac{\partial g(\theta)}{\partial b_\mu} &= \frac{\partial \ln \mathcal{L}(\theta)}{\partial b_\mu} + \gamma \cdot \text{sgn}(b_\mu)\end{aligned} \tag{13}$$

The value of γ needed to properly balance between approximation and generalization errors is a hyperparameter of the model that needs to be chosen.
Applying L1 regularization often causes many of the weights to become zero whilst allowing a few of the weights to grow quite large. This can make it easier to interpret the weights role for digits generation.

2.3. Initialization of weights and biases

Proper initialization of parameters is crucial for efficient training. The bias of the visible units is initialized using Hinton's approach [3]. The activation probability of a visible neuron in RBM is given by the first equation in Eq. 9. If the weights are initialized to small random values chosen from a zero-mean Gaussian ($w_{ij} \approx 0$), then the activation probability of a neuron is $P(x_i = 1) \approx \sigma(a_i)$. The idea is to initialize the weights such that the activation probability $P(x_i = 1) = p_i$, where p_i is the fraction of training data in which the i-th unit is active. From Eq. 9 and 10, this entails

$$a_i = \log(\frac{p_i}{1 - p_i}) \tag{14}$$

This method prevents the network from starting from an uninformative state, facilitating faster convergence. In fact, this reduces the chance to end up in local minima of the negative log-likelihood, far from the absolute minimum we are interested in.
The weights are initialized using a Gaussian distribution with standard deviation proposed by Glorot and Bengio [4], which scales with the size of the layers:

$$\sigma = \frac{2}{\sqrt{L + D}} \tag{15}$$

2.4. Log-likelihood estimation

We denote the log-likelihood per data point $\mathbf{x}$ as $\ell_\theta(\mathbf{x}) = \ln p_\theta(\mathbf{x})$. For an RBM, this can be written as:

$$\ell_\theta(\mathbf{x}) = \underbrace{\ln \sum_{\{\mathbf{z}\}} e^{-E(\mathbf{x},\, \mathbf{z})}}_{\text{free energy term}} - \underbrace{\ln \sum_{\{\mathbf{x}'\}} \sum_{\{\mathbf{z}\}} e^{-E(\mathbf{x}',\, \mathbf{z})}}_{\text{partition function Z}} \tag{16}$$

This formulation consists of two main components: the first term accounts for the free energy, which captures the contribution of the hidden units given a visible state $\mathbf{x}$; the second term represents the contribution coming from the partition function Z, which must be computed summing over all possible configurations of visible and hidden units. Given an RBM with D visible units and L hidden units, the number of possible configurations scales as 2^{D+L}, making exact computation unfeasible for large systems due to the exponential complexity.

However, we can use the energy function as a helpful formula (Eq. 1). For the partition function term, we can rewrite the Boltzmann weight of a given configuration $(\mathbf{x}, \mathbf{z})$

$$e^{-E(\mathbf{x},\, \mathbf{z})} = \underbrace{\prod_{\mu} e^{b_\mu z_\mu}}_{G(\mathbf{z})} \prod_i e^{H_i(\mathbf{z}) x_i} \tag{17}$$

where $H_i(\mathbf{z}) = a_i + \sum_\mu w_{i\mu} z_\mu$. For the usual Bernoulli case, we can then obtain the expression:

$$\ln Z = D \ln q + \ln \left[\sum_{\mathbf{z}} G(\mathbf{z}) \prod_{i=1}^{D} \frac{1 + e^{H_i(\mathbf{z})}}{q} \right] \tag{18}$$

where $q \approx \text{avg}_i(1 + e^{H_i(\mathbf{z})})$ is a stabilization constant introduced to prevent numerical overflows in the product $\prod_i (1 + e^{H_i(\mathbf{z})})$. By factoring out q, we keep the terms near unity, improving numerical stability without affecting the correctness of the result. The correction term $D \ln q$ in Eq. 18 accounts for the rescaling.
This formulation allows an efficient approximation of the single log-likelihood, which is then averaged over a fraction of the dataset to estimate the overall likelihood of the model.

The direct computation of the free energy term can result in numerical overflows. To avoid this and to ensure numerical stability, we use the following trick

$$\ln \sum_i e^{x_i} = a + \ln \left(\sum_i e^{x_i} - a \right) \tag{19}$$

where $a = \max\{x_i\}$ serves as a stabilizer. In the implementation: we first compute all logarithmic terms (representing the contributions of different hidden states), we then identify the maximum log-term to use as a reference shift and finally we compute the sum of exponentials of the adjusted terms and take the logarithm.

3. STUDY OF THE PERFORMANCE FOR DIFFERENT HYPERPERAMETERS

3.1. DEPENDANCE ON THE NUMBER OF HIDDEN UNITS AND CD STEPS

3.1.1 Likelihood evolution and stability

We want to examine how the model performs for a different number of hidden neurons L and steps N_t considered in CD during the training process.
Here we use Root Mean Square Propagation (RMSprop) as maximization algorithm (with parameters

β=0.9, ϵ=10^{-4}) and fix the regularization parameter to $\gamma = 10^{-3}$. For computational feasibility we will not consider a full grid-search, but when examining the dependance for variable L, N_t will be fixed and vice-versa, meaning we are assuming little correlation between the two parameters.

Firstly, we examine the likelihood behaviour during the whole training. We consider single runs for training and single log-likelihood estimations for each parameter. From Fig. 3 we notice, as expected, that the likelihood increases during a transient phase, after which it appears to become more stable.

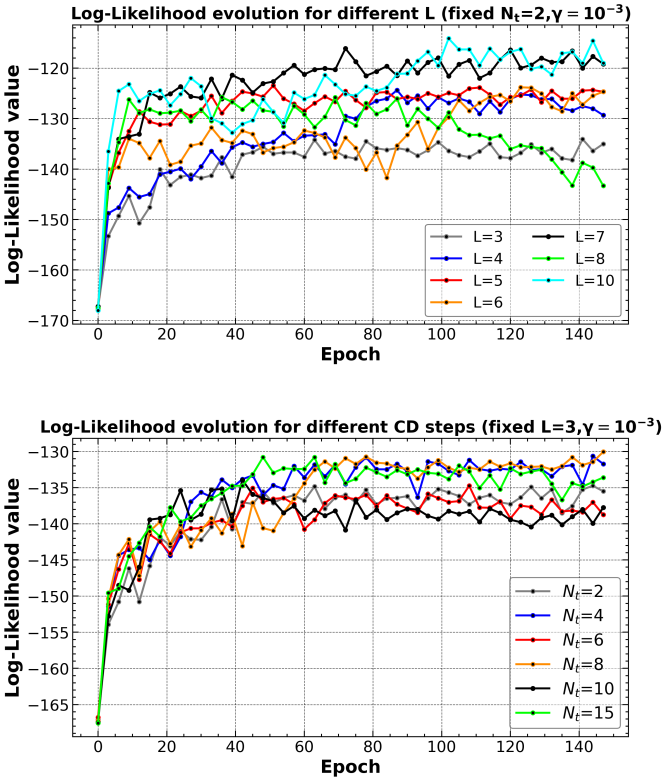

FIG. 3: Evolution of the likelihood during training (single runs): we can observe a transient with a steady increase and then a more stable behaviour.

In order to verify this rigorously, we focus on the last 10 epochs and perform a χ^2 test for a uniform distribution (CL = 95%, $\chi^2_0 \approx 16.9$). We also consider the average of 10 iterations for each likelihood estimation to reduce the randomness due to sample choice. Results are presented in Fig. 4.
In the majority of the cases considered, the test would suggest that actual stability is not reached ($\chi^2 > \chi^2_0$). This is particularly evident for the highest values of hidden units ($L = 8, 10$) in which we observe high fluctuations of the likelihood, but it is not clear the dependance of the stability on L or N_t. Thus in the following, we consider as error of the associated mean value over the last epochs, the error of the mean in case of compatibility, and the standard deviation (error of "single measurement") in case of non compatibility.

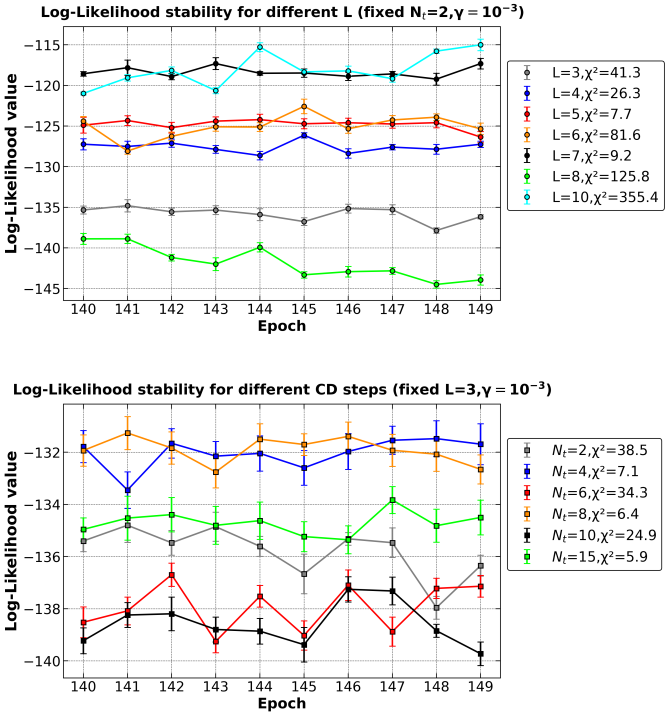

FIG. 4: Analysis of likelihood stability over the last 10 epochs (10 iterations for each likelihood estimation). A χ^2 test for a stationary distribution has been performed (CL = 95%, $\chi^2_0 \approx 16.9$).

3.1.2 Study of the variability for different training sets and inizializations

We analyze the variability of the likelihood for different training runs. In order to compare the behaviour of different hyperparameters, we have to consider the average performance of that model over many different training sets and initial conditions.
In Fig. 17 (see Appendix), we show the evolution over the last 10 epochs for 5 different runs of the same models ($L = 3, 4, 5, 6, 7, 8$ with $N_t = 2$). For each run, we calculate the weighted average over the last 10 epochs. Then, for each L we determine the mean and standard deviation σ_L of the log-likelihoods over the different runs. This σ_L is an estimate of the error associated with a single run for the likelihood value after the transient. In the following, when multiple (N) runs will be available, we determine the error of their mean as uncertainty estimation ($\sigma/\sqrt{N}$). Instead, when only single runs will be available, we associate as error of the model performance the previous σ_L found using the corresponding L.

Fig. 5 shows an evident and sudden increase in the standard deviation of the likelihood for $L = 8$. It is reasonable to think that more complex models will be more sensible to the training set and weights initialization. However, we do not have sufficient data to spot the clear dependence of $\sigma_{log(L)}$ from L since training and especially likelihood estimation are very computationally heavy for higher numbers of hidden neurons.

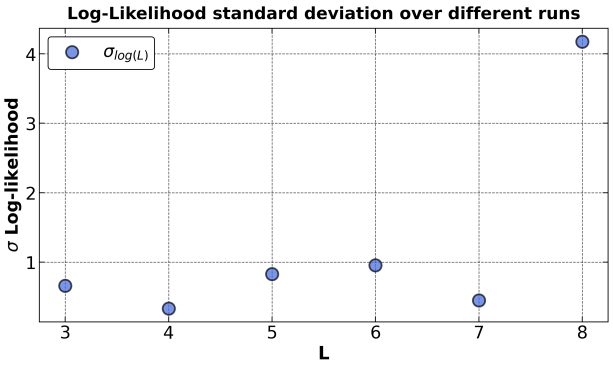

FIG. 5: Standard deviation of the likelihood value after the transient across 5 different training runs for each L.

3.1.3 Dependance on L and N_t: results

In Fig. 6 we present the results: we run 5 iterations in the case of variable L, while we consider a single run each in the case of variable N_t since the $\sigma_{L=3}$ associated is already sufficiently small. We notice a clear trend of improvement for the likelihood with an increasing number of hidden units up to $L = 5$, after which the likelihood seems to stabilize for $L = 6$ to suddenly spike again for $L = 7$. The important uncertainty associated with the estimate for $L = 8$ prevents us from drawing a clear conclusion regarding its behavior with respect to $L = 5, 6$, but it appears clear that it is not the optimal choice. Due to computational constraints we did not explore multiple runs for configurations with a higher number of hidden units. For similar reasons, for the subsequent analysis we will fix the number of hidden units to $L = 5$, which was identified as the second-best configuration. On the other hand, we observe no significant correlation between the log-likelihood values found and the number of CD steps chosen. We will use $N_t = 4$ for the following analysis, as this choice yields an almost identical performance to that of $N_t = 8$, while being computationally less demanding.

3.2 DEPENDANCE ON THE REGULARIZATION FACTOR γ

We study how the performance of the network changes for different L1 regularization factors:

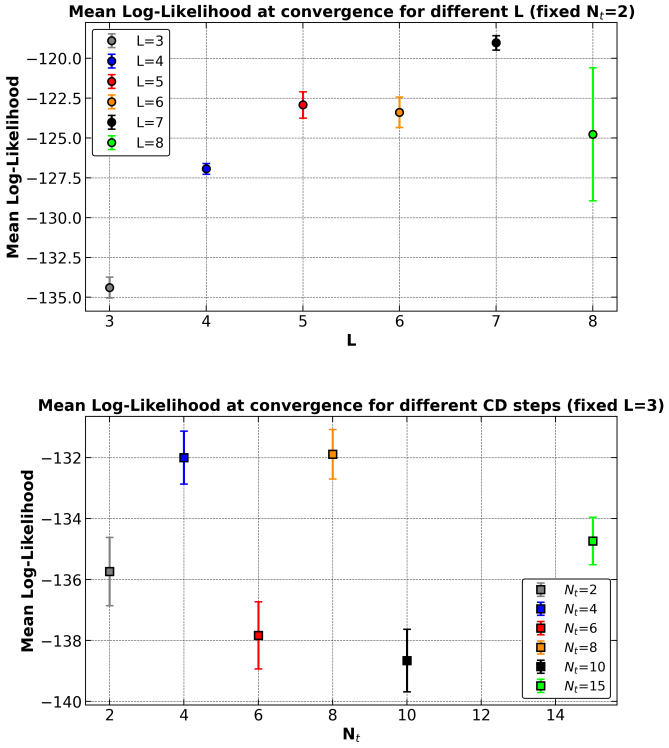

FIG. 6: (a) Mean log-likelihood values over 5 runs for different numbers of hidden units L. (b) Log-likelihood values for different numbers of steps in CD (single runs).

$\gamma = 0.5, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-4}$ in the RMSprop algorithm for likelihood maximization. We perform 5 runs for each choice of γ. In Fig. 7 we present the results. We can spot a clear parabolic trend with a peak around $\gamma = 10^{-3}$, which appears to be the preferable choice, even though the results obtained for $\gamma = 10^{-2}$ are still compatible within the error associated. We can conclude that including regularization in the algorithm is preferable, but this needs caution because excessive values of γ lead to significantly worse performances.

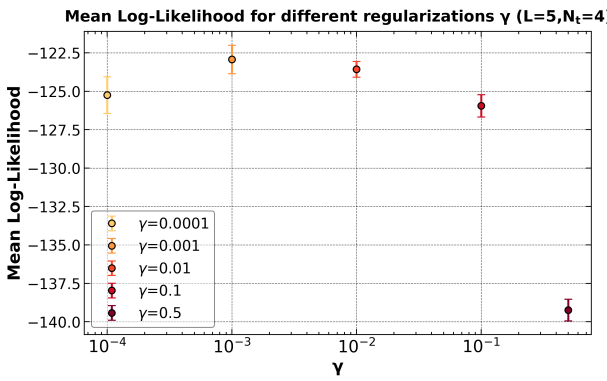

FIG. 7: Mean log-likelihood values over 5 runs for different regularization parameters γ.

3.3 DEPENDANCE ON THE CHOICE OF MAXIMIZATION ALGORITHM

We compare the likelihood after transient obtained using standard SGD and RMSprop algorithms for training, fixed the other hypeparameters to the previously chosen values $L = 5$, $N_t = 4$, $\gamma = 10^{-3}$. Again, we consider the average over 5 runs for each method. Results are presented in Fig. 8 and 9.

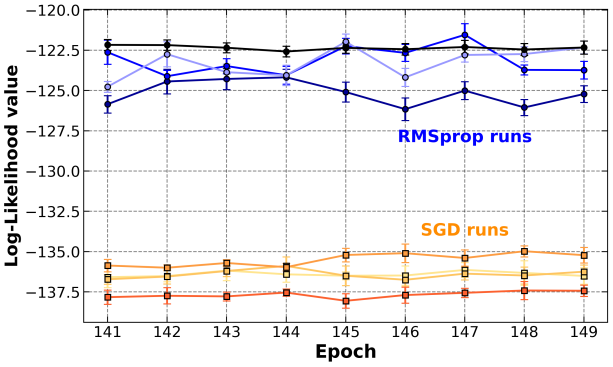

FIG. 8: Log-likelihood values after transient for 5 runs of SGD and RMSprop algorithms.

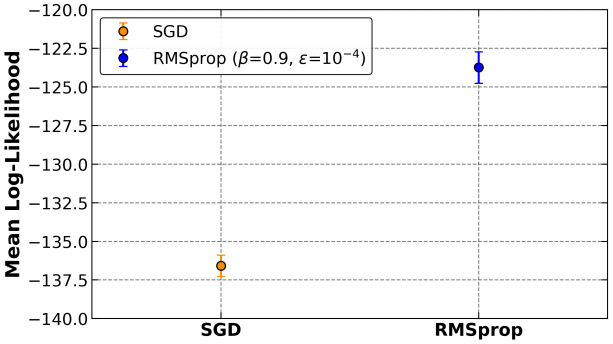

FIG. 9: Mean log-likelihood values over 5 runs using SGD and RMSprop algorithms.

We notice a clear separation between the log-likelihood values of two methods, with RMSprop performing significantly better over the same number of epochs.
This is consistent with expectations, as RMSprop is an optimized version of the simpler SGD algorithm. While in standard SGD the learning rate increases linearly with each epoch and is applied uniformly across all weight components, RMSprop enhances the convergence speed and stability of the model during training by using the moving average of the squared gradients to scale the learning rate individually for each weight component.

3.4 POTTS AND SPINS MODELS

In this section we investigate the performance that can be achieved using the Potts model, in which only one of the L hidden neurons can be active at a time (one-hot encoding of the hidden states), and the Spins model, in which all neurons can assume state $x, z = \{\pm 1\}$.
In particular, we analyze how the likelihood varies for different values of L, fixed $N_t = 4$ and $\gamma = 10^{-3}$, since we expect this to remain the most important hyperparameter. Averaging over five runs for each case, we compare the results with those obtained previously for the standard Bernoulli model, as shown in Fig. 10. Since log-likelihood estimation is faster in the Potts model, we also consider the case $L = 10$.

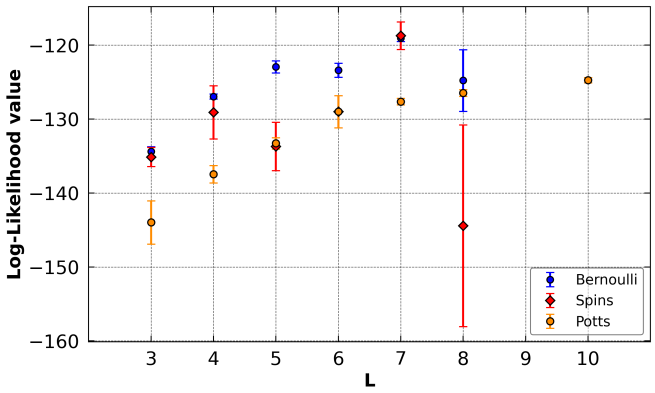

FIG. 10: Comparison between Bernoulli, Potts and Spins model of mean log-likelihood values over 5 runs for different L.

We observe a clear trend of improvement for the log-likelihood with increasing L with Potts, although it consistently performs significantly worse than the corresponding L value in the usual Bernoulli case. This is to be expected, as the Potts model is less complex, with only L possible configurations for the hidden units compared to the usual 2^L. As a result, its performance will be inherently worse then the standard case and crucially influenced by the number of hidden neurons, especially for low L.
An interesting observation is that, in the Potts case, the variability of the likelihood across different training runs appears to be significantly reduced for higher L. This is surely advantageous, as it contributes to more consistent and reproducible results.

Similarly, also the Spins model fails to outperform the corresponding Bernoulli case and, furthermore, exhibits a significantly higher variability of the likelihood across different trainings. This can be explained by the fact that in the Spin model the RBM is forced to properly tune all weights and cannot exploit the neutral state (i.e. all neurons off) in the learning process.
In fact, as further discussed later, in the Bernoulli case the bias term can effectively represent a digit per se, and thus the neutral state becomes useful. The weights associated with this neutral state do not require fine-tuning,

simplifying the learning process and contributing to better and more stable results.

4. EXPLORING HIDDEN STATES ACTIVATION

Our aim in this section is to check which hidden units are activated for a selected RBM. In the following analysis, we choose to focus on the best model with $L = 5$ found.

Each different digit activates specific hidden states with its own probability. For each digit, we can then derive the PDF (probability density function) of activation of the hidden states. This enables us to enlighten the correlations between the activation of the single L hidden units.

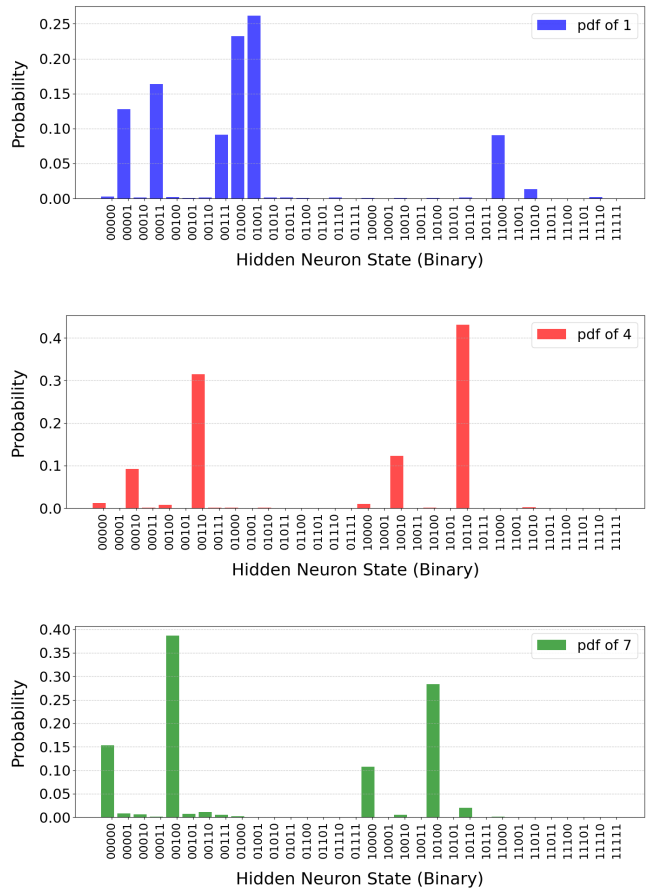

FIG. 11: Histograms representing the PDFs of hidden activations for (a) digit 1, (b) digit 4, (c) digit 7.

In Fig. 11 we show the activation probabilities of all possible hidden states per each digit (at the end of training). We observe that each digit activates a distinct set of hidden states, with minimal overlap or correlations between them. This suggests that the RBM effectively allocates its limited hidden state configurations to focus on generating a specific digit and not multiple digits simultaneously.

We can also note that digit 1 activates a fair amount of different states with high probabilities, while digits 4 and 7 only activate four states with higher probabilities. This suggests that the machine has learnt compact representations for 4 and 7, while different representations for digit 1 have been found. To better visualize the representations of the digits, in Fig. 12 we generate the images of the activations of the 5 most relevant hidden states.

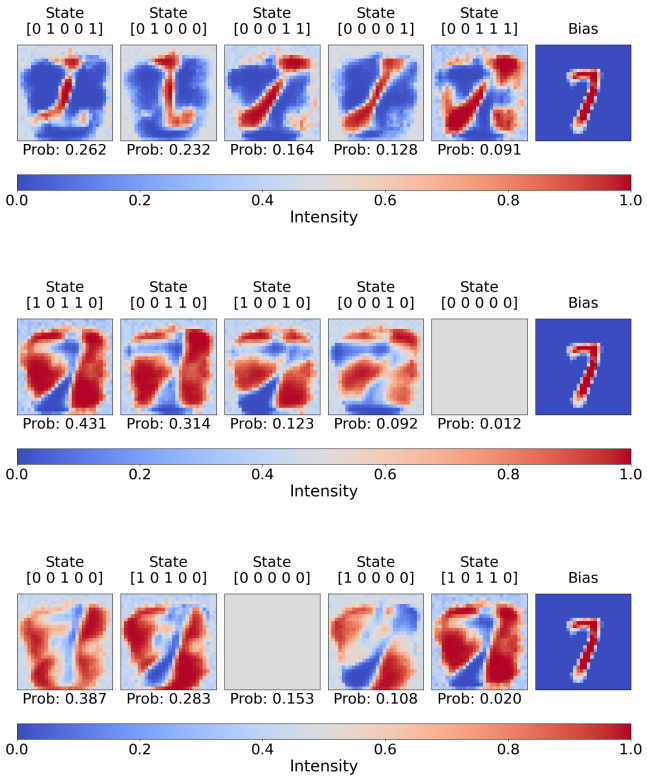

FIG. 12: Visualization of the hidden states activated by (a) digit 1, (b) digit 4, (c) digit 7.

The red areas are activated with high intensity, the blue ones with low. The bias is also represented, and we notice that its shape resembles the one of a 7. This justifies why one of the most represented states by the digit 7 is a neutral state (the grey one): the bias alone well represents digit 7, so in most cases the bias and the first state are sufficient for its generation. As far as the digit 1 is concerned, the hidden states all visually resemble the shape of a 1, consistently with what found above. The same is not true for the digit 4: hidden states do not show explicit 4 patterns and, taken one by one, do not really resemble any digit.

A possible check can be performed on the single units:

one unit may be activated by different hidden states. We compute then the correlation matrices between the probabilities of activation of the single neurons. Neurons have been indexed from 0 to 4. In Fig. 13 we show the correlation matrices.

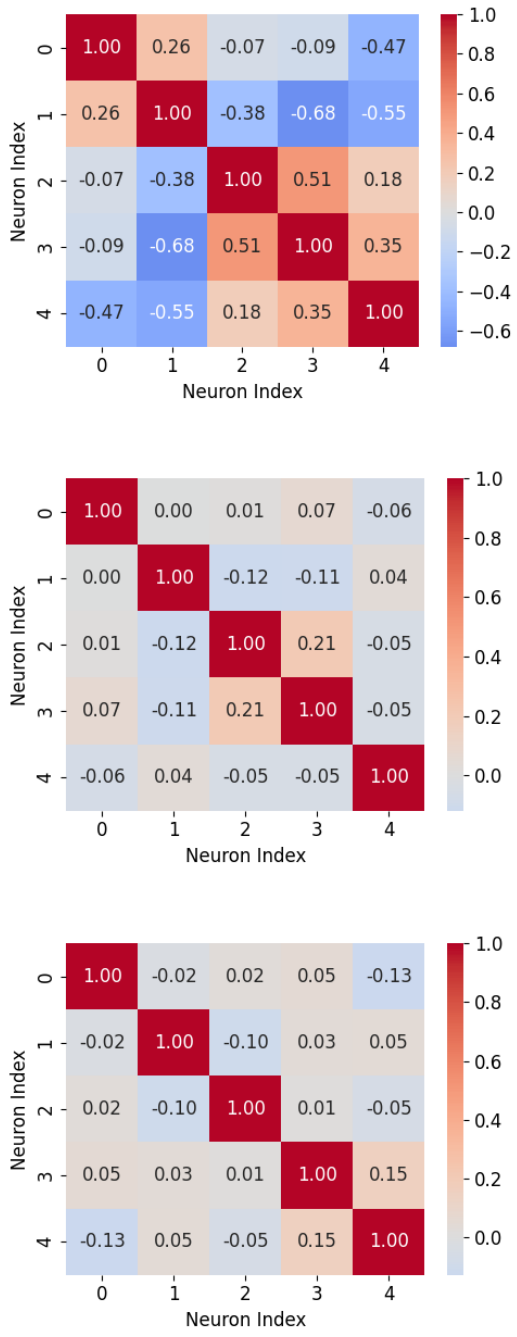

FIG. 13: Correlation matrices of activation probabilities of each neuron by (a) digit 1, (b) digit 4, (c) digit 7.

It is easily observable that, for digit 4 and 7, there are almost no relevant correlations between different neurons, while for digit 1 high correlation is present, in particular between neuron 1 and the others. This leads us to conclude that the different representations for digit 1 are more consistent in the training set, leading different graphical features to appear together. Instead, 4 and 7 can be represented in a variety of shapes, leading the RBM to identify different local graphical features that are not correlated to each other.

We can check the effectiveness of the hidden states representation by testing the generative power of the RBM. This is done by giving as input a random image from the training dataset and running the CD algorithm for a fixed number of steps with the trained model. In Fig. 14 we can see the evolution of the representation of each digit through the contrastive divergence steps. The first represented digit, in red, is the random image chosen for generation. The latter images follow a color scale to represent the 40^{th} iterations of CD. We observe that the RBM is able to generate in a satisfactory way the digits it was trained on, with all digits being clearly recognizable.

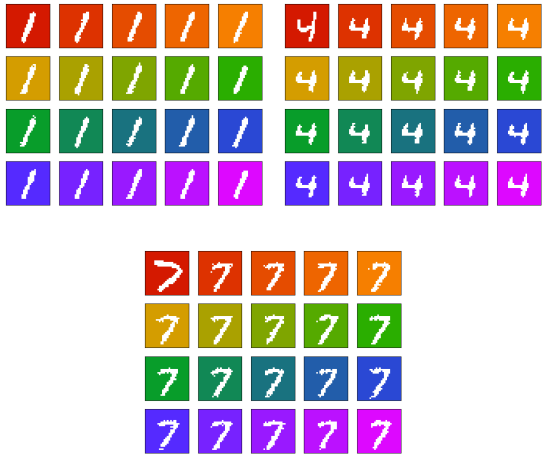

FIG. 14: Visualization of the generation of digit 1, 4 and 7: the evolution over the CD iterations is represnted with different colours.

5. EXPLORING ENERGY BARRIERS

We noted that CD algorithm does not switch hidden representations very much and continues to give back the same digit over and over. This quasi-ergodicity of CD coauld be justifies by a high energy barrier separating the typical $(\mathbf{x}, \mathbf{z})$ configurations when $\mathbf{x}$ are two different digits. We expect that, forcing a transition from a 1 to another digit, the energy of the intermediate states will be higher than the energy of the base states. To test the existence of this energy barrier, we train a RBM with standard CD and fix the hyperparameters to the

best ones found above. After the training we are able to compute the energy of a state $(\mathbf{x}, \mathbf{z})$, given by Eq. 1. We then need a way to force the transition between different digits. This can be achieved by modifying the defintion of the energy formula with input data $\mathbf{x_t}$:

$$\mathbf{x_t} = (1 - \alpha)\mathbf{x_1} + \alpha\mathbf{x_2} \tag{20}$$

where $\mathbf{x_1}$ and $\mathbf{x_2}$ are images relative to different digits, and α is a parameter that can be tuned in the range $[0, 1]$ in order to get intermediate images between $\mathbf{x_1}$ and $\mathbf{x_2}$. We compute the energy of 1000 transitions from the lowest energy digit (1) to the other digits. We perform this calculation using the weights of two different epochs: one from the start of the training and the very last one. The resulting plots are shown in Fig. 15.

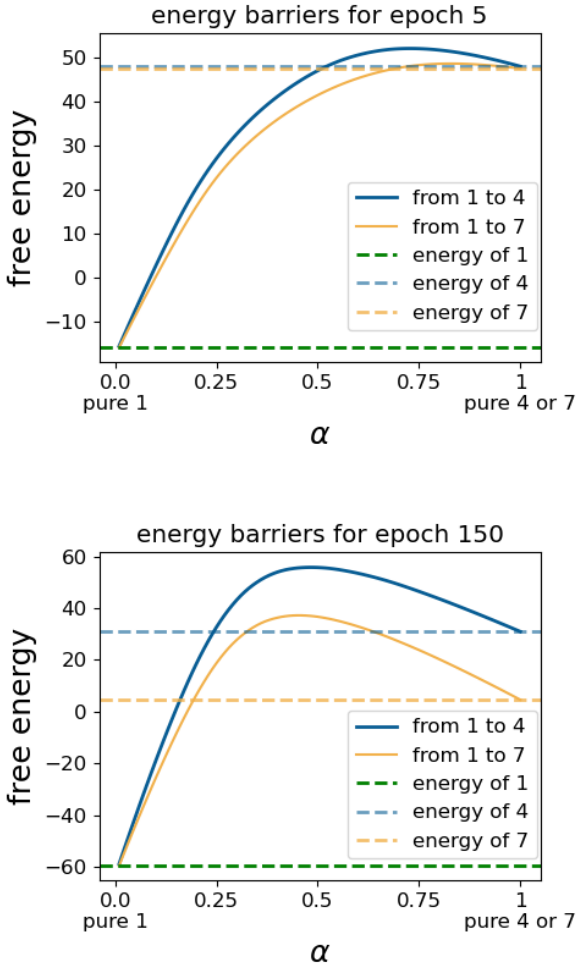

FIG. 15: Visualization of the energy barriers relative to the transitions from 1 to 4 or 7: (a) at the beginning of the training and (b) at the end.

We observe that in Fig.15a, relative to an epoch at the start of the training, the energy gap between 4, 7 at the highest energy is relatively small. At the start of the training the RBM can still overcome the energy gaps, but once it reaches the "valley" corresponding to the 1, it is very hard for it to escape from it. This becomes progressively more difficult as the energy gap increases during training, as seen in Fig.15b. We observe that, given these digits, the representation of 1 is the less costly for our RBM, while 4 and 7 require more energy.

We then propose a variation on the traditional CD algorithm. Our objective is to get different digits to have the same energy. We can achieve this by adding a bias to 4 and 7 digits. The algorithm is then run and every time the RBM is trained on a 4 or a 7 a fixed value of bias is added to the weights. The results, in terms of energy, can be seen in Fig. 16.

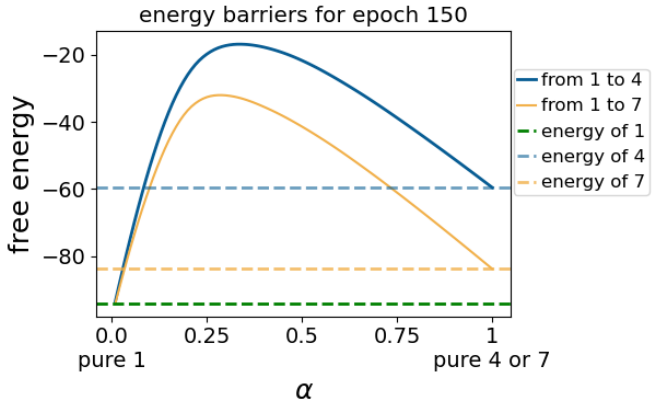

FIG. 16: Visualization of the energy barriers relative to the transitions from 1 to 4 or 7. A bias has been added to 4 and 7 with the intent of lowering the energy barrier between digits.

We can see that the energy for 4 and 7 is now lower and more compatible with the energy of 1. Despite this, the intermediate configurations still has a very high energy, and the gap is hardly overcome with standard CD. A different approach to overtake the gap can be tried adding a temperature parameter T to the computation of the weights. Multiplying the visible biases by a value $T > 1$ increases exploration during training, and can allow the RBM to overcome energy gaps. We explored this option, but did not obtain significant results.

CONCLUSIONS

We found satisfactory results for the best RBM model trained: the generated digits (1,4 and 7) are easily identifiable and distinguishable.

Our analysis concludes that, for the specific task

considered, standard Bernoulli model outperforms both the Potts and Spins variants. In particular, the most influential hyperparameter of the model is the number of hidden units L, generally resulting in better performance for higher values of L, but with a fast increase of the computational time needed for training. We found that L=7 yields the best performance within the considered range. On the other hand, the number of steps in Contrastive Divergence N_t does not appear to significantly affect the model's performance for the task examined, with N_t=8 yielding the best results among the considered configurations.

As expected, we confirm that RMSprop training algorithm performs much better than standard SGD and is to be preferred; we also conclude that the choice of a correct L1 regularization factor γ modestly influences results, with $\gamma = 10^{-3}$ being the preferable choice and large values ($\gamma > 0.1$) to be especially avoided.

With regard to the activation of the hidden units for each chosen digit, we observe that the RBM allocates distinct sets of hidden states to represent a specific digit, without overlapping. In this way the Machine appears to optimize the available model complexity and its generative capacity. We can also conclude that, since digit 1 is more consistently represented in terms of shapes in the training set, while 4 and 7 exhibit more distinct representations, the latter require the RBM to identify different local graphical features that are uncorrelated with each other. In general, we observe that the RBM is able to generate efficiently all digits it has been trained on, but with a tendency to favour the representation of the digit 1, since 4 and 7 are more energy greedy.

At last, we observe that the energy gap between the digits becomes significantly more relevant as the RBM training progresses, and it is hardly surmountable with the use of standard CD. This provides an explanation for the fact that we do not usually observe significant changes in the hidden states representation during the training. We found that the introduction of a digit-specific bias can even out the energies of the different digits, but this still does not manage to flatten the energy curves.

[1] P. Mehta, M. Bukov, C. H. Wang, A. G. R. Day, C. Richardson, C. K. Fisher, and D. J. Schwab, Physics Reports **810**, 1 (2019).
[2] A. Decelle and C. Furtlehner, Chinese Physics B **30**, 040202 (2021).
[3] G. E. Hinton, Neural Networks: Tricks of the Trade: Second Edition **Second Edition**, 599 (2012).
[4] X. Glorot and Y. Bengio, Journal of Machine Learning Research **9**, 249 (2010).

APPENDIX

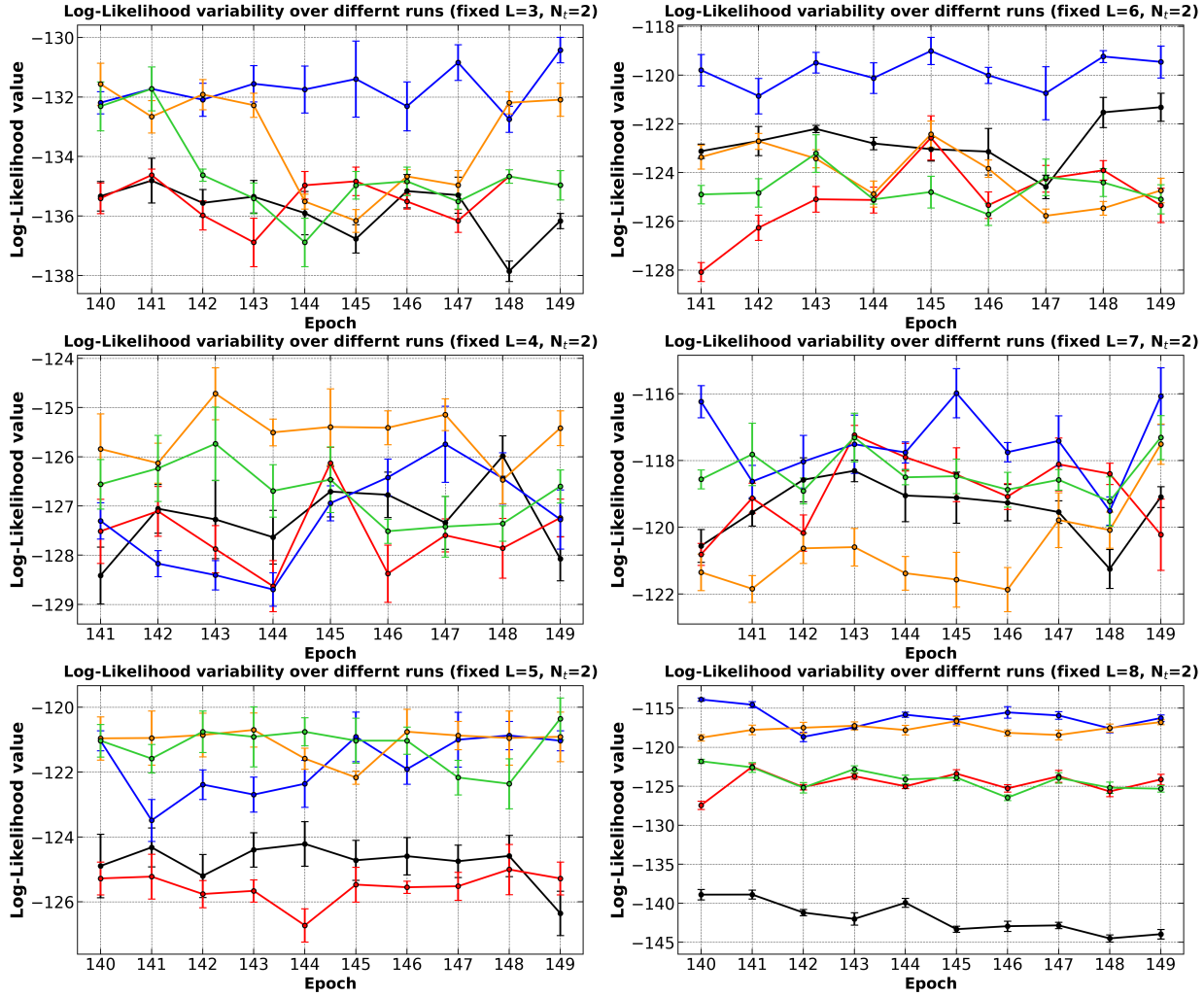

FIG. 17: Log-likelihood values after transient for 5 runs of the same models (variable L).