Unsupervised Learning and Log-likelihood Analysis with Restricted Boltzmann Machines

Elias Maria Bonasera, Alberto Casellato, Nicola Garbin, and Francesco Pazzocco

We present a comprehensive study on the use of Restricted Boltzmann Machines (RBMs) for unsupervised learning of handwritten digit features from the MNIST dataset. Through experiments with a custom Python implementation, we focus on the RBM architecture, the training via Contrastive Divergence, and extensive hyperparameter tuning. We describe the experimental setup, the log-likelihood estimation method, and evaluate different configurations. The results highlight both the strengths and limitations of the approach, suggesting directions for future improvements.

INTRODUCTION

Unsupervised learning is essential for extracting features from unlabeled data, and probabilistic graphical models (PGMs), such as Markov Random Fields (MRFs), provide a framework for this task. Restricted Boltzmann Machines (RBMs) are a specific type of MRF with a bipartite structure (a visible and a hidden layer with no intra-layer connections). They are important both theoretically and for practical applications for their ability to model complex distributions and for learning efficient representations of intricate data relatively fast[1–3].

In this work, we present a complete implementation of a Restricted Boltzmann Machine (RBM) and analyze its behavior through systematic experiments. The paper is organized as follows: Section 1 presents the theoretical foundation of RBMs, Section 2 describes the experimental methodology and implementation specifics, Section 3 shows the experimental results accompanied by graphical analysis, and Section 4 discusses the findings, draws conclusions, and suggests avenues for future research. Supplementary equations and technical details are provided in the Appendix.

1. THEORY

Restricted Boltzmann Machines

RBMs are generative models consisting of a visible layer, representing the input data, and a hidden layer that captures latent features.

Mathematically, RMBs consist in a set of D binary visible units i of state $\{v_i\}_{i=1,\dots,D}$ symmetrically connected to a set of L binary hidden units μ of state $\{h_\mu\}_{\mu=1,\dots,L}$. The continuos weight $w_{i\mu}$ quantifies the strength between unit i and unit μ (see Figure 1). [1, 2].

The energy of a configuration $(\mathbf{v}, \mathbf{h})$ is defined as

$$E(\mathbf{v}, \mathbf{h}) = -\sum_i a_i\, v_i - \sum_\mu b_\mu\, h_\mu - \sum_{i,\,\mu} v_i\, w_{i\mu}\, h_\mu \qquad (1)$$

where a_i and b_μ are the biases for the visible and hidden units, respectively, and $w_{i\mu}$ denotes the weight between them. No intra-layer connections exist, which simplifies the energy function and learning dynamics.

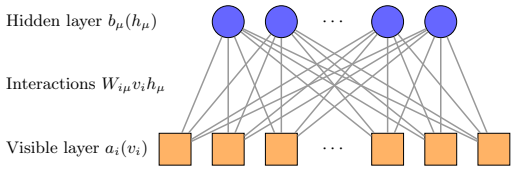


FIG. 1: Bipartite structure of a Restricted Boltzmann Machine. Visible units (bottom) are connected to hidden units (top) with no intra-layer connections.

The probability of a hidden unit being active is computed using the logistic sigmoid for both visibile and hidden units:

$$p(h_\mu = 1|\mathbf{v}) = \sigma\left(b_\mu + \sum_i v_i\, w_{i\mu}\right) \qquad (2)$$

$$p(v_i = 1|\mathbf{h}) = \sigma\left(a_i + \sum_\mu h_\mu\, w_{i\mu}\right) \qquad (3)$$

The training consists in minimizing the energy's $\boldsymbol{\theta}$ parameters, typically using the Contrastive Divergence (CD) algorithm. This algorithm approximates the gradient of the log-likelihood by alternating Gibbs sampling between the visible and hidden layers [1–3].

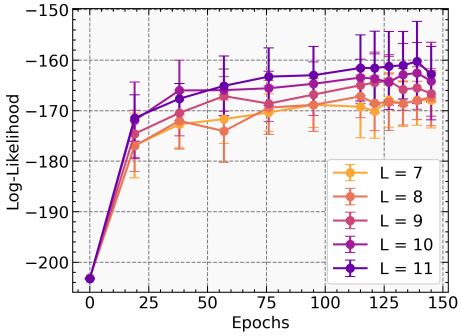
2. METHODS

Experimental Setup and Preprocessing

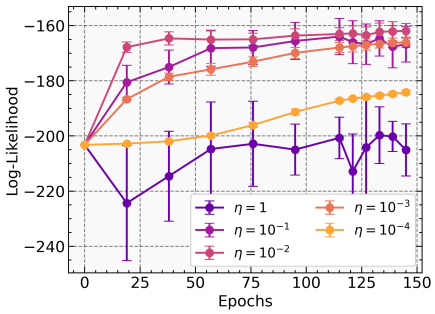
The MNIST dataset consists of 70,000 images of handwritten digits.

Each image is flattened into a 784-dimensional vector and binarized using a threshold of 0.5 (Figure 2).

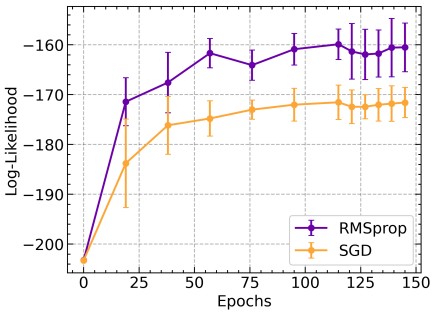
FIG. 2: Example of the binarized MNIST dataset used during training.



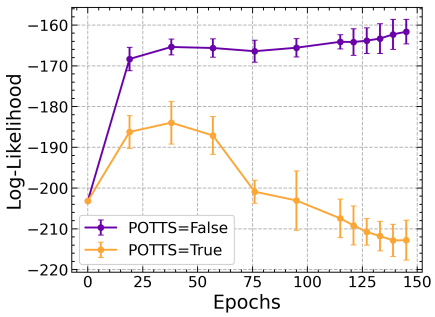
(a) Curves for different hidden units (L).



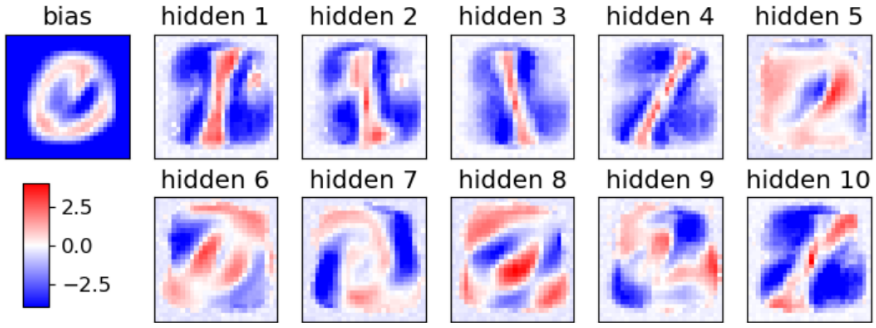
(b) Curves for different learning rates (η).



(c) Cruves for different optimizers.



(d) RMSprop profiles Potts On/Off



(e) Hidden Layers Weights and Bias Visualization.

FIG. 3: **RBM tuning and Feature Extraction.** (a–d) Log-likelihood curves over training epochs for various RBM configurations (different hidden units, learning rates, optimizers, and the presence/absence of POTTS regularization) using Bernoulli encoding. Fig. (e) shows how the model captures distinct digit features and extracts meaningful patterns from the three digits training set. The visualizations under Bernoulli encoding appear well structured and intense, reflecting higher-magnitude weights.

We focused the training on the numbers 0, 1, 2 of the dataset using two encoding schemes:

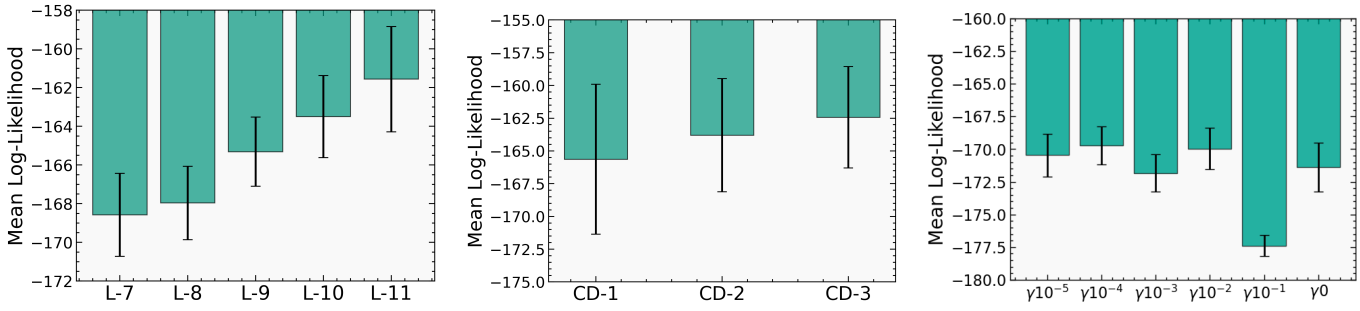
- **Bernoulli encoding:** pixel values in $\{0, 1\}$.
- **Spin encoding:** pixel values in $\{-1, +1\}$.

The RBM is implemented in a Python class (`RBM`) with configurable hyperparameters.

Hyperparameter Tuning

Hyperparameters are systematically varied to assess their effect on model performance [1–3]. These include:

- *Number of hidden units (from 3 to 13)*: Determines the model's capacity. Too few may lead to underfitting, while too many could cause overfitting.
- *Contrastive Divergence steps ($k = 1, 2, 3$)*: Affects the accuracy of gradient estimation; lower k may introduce bias, whereas higher k could improve approximation but slow down training.
- *Learning rate (η)*: Sets the step size for updating parameters (Eq. 6 and Eq. 7). A value that is too high may cause instability and divergence, while a value that is too low can slow convergence and risk getting trapped in suboptimal minima.
- *L1 regularization strength (γ)*: Coefficient for the weight decay (Eq. 5). It penalizes large weights to promote sparsity and prevent overfitting. However, an excessively high γ may lead to underfitting, whereas too low a value might not sufficiently regularize the model.
- *Optimizer type (SGD or RMSprop)*: Influences how weights are updated, impacting convergence speed and the ability to escape local minima.
- *Data encoding (Bernoulli or Spin)*: Alters the energy function and conditional probabilities, thereby affecting how well the model captures underlying data characteristics. Spin encoding, in particular, has been associated with instability or lower performance.
- *POTTS activation (on or off)*: Controls whether the hidden layer follows a one-hot activation pattern, meaning that only one of the hidden units is activated each iteration. POTTS activations can



(a) Final Log-Likelihood for L (b) Final Log-Likelihood for CD steps (c) Final Log-Likelihood for γ

FIG. 4: Final mean log-likelihood across training runs for different hyperparameters using Bernoulli encoding. **(a)** Effect of varying the number of hidden units L ($L = 10$ as a good compromise between log-likelihood and standard deviation), **(b)** Effect of varying the number of Contrastive Divergence (CD) steps ($CD = 3$ chosen despite all values being comparable), and **(c)** Effect of varying regularization strength γ ($\gamma = 10^{-4}$ providing the best performance).

reduce redundancy, but they also lead to instability and degraded performance compared to the standard activation scheme.

For each configuration, training is repeated over multiple runs (each executed 10 times) to ensure statistically robust results.

Log-Likelihood Estimation

The performance of the RBM is evaluated using the log-likelihood:

$$\mathcal{L} = \frac{1}{M} \sum_{m=1}^{M} \log \left(\sum_{\mathbf{h}} e^{-E(\mathbf{v}^{(m)}, \mathbf{h})} \right) - \log Z \qquad (4)$$

where Z is the partition function. For further details, see Section 5.

Training and Optimization Procedure

To update the weights and biases during training, we use two different gradient-based optimization methods: SGD and RMSprop. Their update equations are provided in the Appendix (Eq. 6 and Eq. 7), using the "+" sign for the gradient ascent, as the objective is to maximize a function that increases monotonically. L1 regularization (Eq. 5), controlled by the coefficient γ, is applied after the gradient ascent step. This allows to both track training progress and to ensure consistent model performance across different hyperparameter settings [1, 2].

3. RESULTS

Steady improvement of log-likelihood

As shown in Figures 3a, 3b, and 3c, the log-likelihood steadily increases during training and saturates around epoch 100. This indicates that the RBM is effectively learning to model the data [2, 3].

CD-1 vs CD-2	CD-2 vs CD-3	CD-3 vs CD-1
$t = 0.256$	$t = 0.236$	$t = 0.464$

TABLE I: t-values for each pair of chosen CD steps.

Effects of CD steps

Figure 4b shows that increasing the number of CD steps from 1 to 3 yields only marginal improvements in log-likelihood. Furthermore, a t-test analysis conducted after 100 epochs shows a high compatibility between the results (see Table I). This suggests that CD-1 provides a good balance between computational efficiency and gradient approximation. However, fewer steps may introduce bias and risk divergence [2, 3]; for this reason we choose to adopt CD-3 in the following tuning phases.

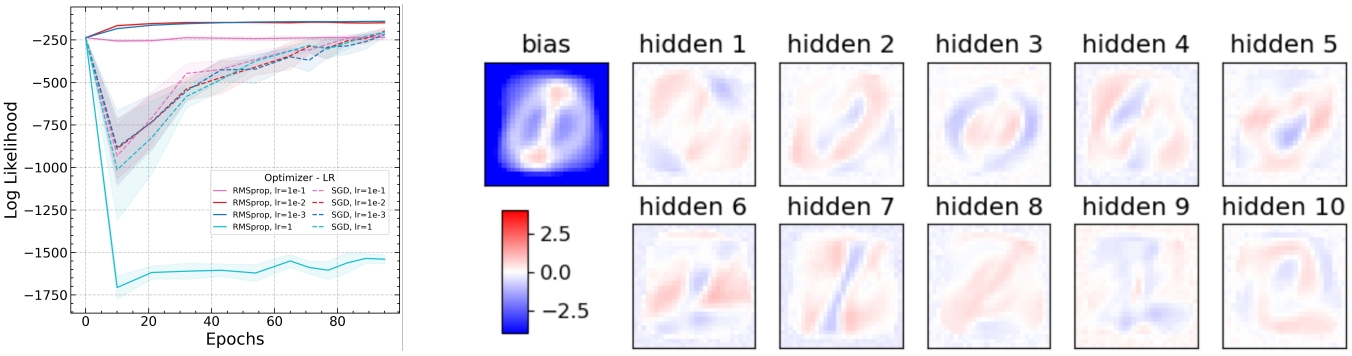
Impact of hidden unit count

With the first preliminar measurement we restricted the interval from $[3, 13]$ to $[7, 11]$. As shown in Figure 3a and Figure 4a, increasing the number of hidden units improves performance up to a point, after which the benefits plateau. This reflects the model's ability to capture data complexity without incurring overfitting when the capacity is appropriately chosen [2, 3].

Effect of Learning Rate and Regularization

Varying the learning rate from 1 to 10^{-4} shows that the log-likelihood is maximized for $\eta = 10^{-2}$ in the Bernoulli encoding (see Figure 3b). This result aligns with theoretical expectations: high learning rates may cause divergence, while lower ones slow convergence and increase the risk to find a local minima [2, 3].

For spin encoding, we get an optimal learning rate of $\eta = 10^{-3}$ (see Figure 5a). A possible explanation is that the one-decade lower value is inherently tied to the encoding type. Since the gradient depends on the product



(a) Tuning process in Spin encoding.

(b) Hidden Layers Weights and Bias for the tuned hyperparameters.

FIG. 5: **RBM Tuning in Spin Encoding.** (a) Variation of the learning rate in RMSprop and SGD revealed the optimal value at $\eta = \gamma = 10^{-3}$ (even if $\gamma = 10^{-2}$ seems to maximize when tuning on the 150 epoch it gets outperformed by the optimal) . (b) In the Spin encoding, the visualizations are more subtle compared to the Bernoulli case, likely due to the presence of negative values (-1) in the representation, which may reduce the contrast in the learned weights. While some residual noise remains visible in the hidden units, the model still performs reasonably well after training.

of visible and hidden units (eq. 7), its magnitude is larger for spin units. This occurs because, in the spin encoding, the –1 values tend to persist and influence other hidden units, thereby slowing adaptation. In contrast, in the Bernoulli encoding, the 0 values help neutralize these effects. Although this hypothesis offers a consistent rationale for the observed differences in learning rates, it remains tentative and not definitively confirmed.

Instead, varying the L1 regularization strength γ gives relatively stable results. We explored values ranging from 10^{-5} to 10^{-1}, including $\gamma = 0$. While performance significantly degrades for $\gamma = 10^{-1}$, the log-likelihood remains comparable for all lower values, with the highest value reached by $\gamma = 10^{-4}$ (see Figure 4c).

Influence of activation and encoding schemes

Our results also reveal that one-hot encoding for hidden units (POTTS) and spin encoding can lead to instability or degraded performance (see Figure 3d), highlighting the importance of selecting appropriate unit types and data representations for stable learning [2, 3].

Post training analysis

As shown in Figure 3e, the model trained with Bernoulli encoding and optimal hyperparameters gives a well-structured and interpretable weights pattern across the hidden units and also for the bias in the visible layer. Distinct features corresponding to each digit are represented, indicating that the RBM model has successfully learned important components of date. The same happens for Spin encoding (Figure 5b), but the learned patterns appear less sharp. This comparison show that Bernoulli encoding is more effective for feature extraction.

4. DISCUSSION AND CONCLUSION

Our experiments confirm that Restricted Boltzmann Machines are capable of learning useful representations of handwritten digits in an unsupervised setting. We obtain a comprehensive view of the model's behavior by evaluating a wide range of hyperparameters in different configurations.

The first key insight is that the log-likelihood increases steadily in the training process, indicating that the model is learning. However, the performance saturates after 100 epochs, suggesting that early stopping strategies may be appropriate. For the Contrastive Divergence steps we have that values from 1 to 3 gives only marginal gains.

Increasing the number of hidden units leads to improved log-likelihood, but this increases the risk of overfitting or redundancy.

For the learning rate there are different optimal values based on the type of encoding schema; lower values work better for spin encoding.

The worse performances were given by setting spin encoding for all the units and one hot encoding (POTTS) for the hidden units.

Finally, while RMSprop offers adaptive learning rates and theoretical advantages, our experiments suggest that it does not consistently outperform standard SGD for the RBM training task, possibly due to the relatively simple architecture and dataset used.

In conclusion, our work shows how sensitive RBMs are as we tune different hyperparameters and make some architectural choices, but also provide a baseline for further research. These include adaptive scheduling of CD steps, deeper architectures (e.g., stacked RBMs), or other estimation techniques (such as Annealed Importance Sampling).

5. APPENDIX

Gradient Update Equations and Regularization

Equation for regularization:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t - \gamma \cdot \eta_t \cdot \text{sign}(\boldsymbol{\theta}) \tag{5}$$

Update equation for SGD:

$$\boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t \pm \eta_t \cdot \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) \tag{6}$$

Update equation for RMSprop:

$$\begin{cases} \boldsymbol{g}_t = \nabla_{\boldsymbol{\theta}} E(\boldsymbol{\theta}) \\ \boldsymbol{s}_t = \beta \boldsymbol{s}_{t-1} + (1-\beta)\boldsymbol{g}_t^2 \\ \boldsymbol{\theta}_{t+1} = \boldsymbol{\theta}_t \pm \eta_t \frac{\boldsymbol{g}_t}{\sqrt{\boldsymbol{s}_t + \epsilon}} \end{cases} \tag{7}$$

Log-Likelihood and Partition Function Approximations

The log-likelihood $\mathcal{L}$ of the data is defined as

$$\mathcal{L} = \frac{1}{M} \sum_{m=1}^{M} \ell_\theta(v^{(m)}) \tag{8}$$

with

$$\ell_\theta(v^{(m)}) = \log \sum_h e^{-E(v^{(m)},h)} - \log Z \tag{9}$$

Here, M is the number of data points and Z denotes the partition function. The energy function $E(v,h)$ is expressed through

$$e^{-E(v,h)} = G(h) \prod_i e^{H_i(h) v_i} \tag{10}$$

where

$$G(h) = \prod_\mu e^{b_\mu h_\mu} \quad \text{and} \quad H_i(h) = a_i + \sum_\mu w_{i\mu} h_\mu \tag{11}$$

In Equation 9, the summation over h runs over all possible states of the hidden layer.

Since directly computing the partition function Z is computationally intractable, we apply approximations and normalization techniques. A more in-depth mathematical analysis about it can be found in [4].

For the Bernoulli encoding, the partition function is manipulated as:

$$Z = q^D \sum_h G(h) \prod_{i=1}^{D} \frac{1 + e^{H_i(h)}}{q} \tag{12}$$

where q is a scaling factor introduced to avoid numerical overflow. For spin encoding, where units assume values in $\{-1, 1\}$, the partition function is approximated by

$$Z = \sum_h G(h) \prod_{i=1}^{D} 2 \cdot \cosh\left(H_i(h)\right) \tag{13}$$

In eq. 13, to ensure numerical stability, we re-elaborate the product introducing a logaritm to avoid overflow. The product becomes via *logarithmic identity*:

$$\log\left(\prod_{i=1}^{N} 2 \cdot \cosh(x_i)\right) = \underbrace{N \cdot \log(2)}_{\text{from } 2^N} + \sum_{i=1}^{N} \log\left(\cosh(x_i)\right) \tag{14}$$

we compute this sum instead of multiplying large values directly. The log(cosh) is manipulated using the identity

$$\log(\cosh(x)) = |x| - \log(2) + \log\left(1 + e^{-2|x|}\right) \tag{15}$$

which avoids overflow and underflow. In Python we used NUMPY.LOG1P for precision. The sum over $\log(2\cosh(x_i))$ becomes:

$$\begin{aligned} \sum_{i=1}^{N} \log(2\cosh(x_i)) &= \sum_{i=1}^{N} [\log(2) + \log(\cosh(x_i))] \\ &= N\log(2) + \sum_{i=1}^{N} \left[|x_i| - \log(2) + \log\left(1 + e^{-2|x_i|}\right)\right] \end{aligned} \tag{16}$$

The $N\log(2)$ term cancels out, leaving:

$$\sum_{i=1}^{N} \left(|x_i| + \log\left(1 + e^{-2|x_i|}\right)\right) \tag{17}$$

That we used to ensure numerical stability.

Author Contributions

All team members contributed equally to the development and execution of the project

[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] M. Bortoletto, *Study of performances for Restricted Boltzmann Machines*, Final dissertation, Università degli Studi di Padova, Dipartimento di Fisica e Astronomia "Galileo Galilei" (2021).

[3] G. E. Hinton, in *Neural Networks: Tricks of the Trade: Second Edition*, edited by G. Montavon, G. B. Orr, and K.-R. Müller (Springer, 2012) pp. 599–619.

[4] M. Baiesi, "Log-likelihood computation for restricted boltzmann machines (rbms)," (2025), unpublished manuscript.