GROUP 10 – Generative Capabilities of Restricted Boltzmann Machines on the MNIST Dataset

Gabriele Bellavia, Luigi Belli, Margherita Lera, and Giovanni Andrea Maida
(Dated: April 8, 2025)

We investigate the generative capabilities of Restricted Boltzmann Machines on a subset of the MNIST dataset, focusing on the digits '7', '8' and '9'. Starting from a pre-built model, we evaluate different architectures and training configurations, using the log-likelihood $\mathcal{L}$ as the performance metric. We find that the score increases with the number of hidden units reaching $\mathcal{L} = -147 \pm 1$ for 8 units (the limit of our computational capacity). By using one-hot encoding on the activation of hidden units, we are able to further increase this limit, finding similar results for a machine with 55 units: $\mathcal{L} = -146 \pm 1$. We are unsure whether this similarity indicates a better tuning requirement or a cap on the log-likelihood. Nonetheless, the latter model shows a better generative ability, being generally able to reproduce the main features of the input data.

1. INTRODUCTION

Restricted Boltzmann Machines (RBMs) are two-layer networks that can learn a probability distribution starting from a set of inputs. Their architecture is composed of a layer of visible units and one of hidden units. Both visible and hidden units can interact with each other but not among themselves. The hidden variables help in expressing sophisticated correlations between observable features without sacrificing trainability.

RBMs are energy-based generative models. These models relate to the idea of assigning an energy value to different configurations of the data: lower energy is associated with more probable configurations. In opposition to discriminative models, generative models aim at generating new samples similar to those found in a training set. For most of the cases, this is accomplished by trying to learn a parametric model for the probability distribution from which data were drawn.

Goals and structure of the review – The main goal of this work is to create an RBM able to learn and reproduce a set of digit images taken from the MNIST database. Our objectives can be summarized in the following outline:

- Analyze the behavior of the log-likelihood of the probability distribution during the training. Compare then the final log-likelihood obtained with different numbers of hidden units.
- Explore the internal representations learnt by the RBM by analyzing which hidden units are activated by each digit and plotting the probability distribution of each hidden state per digit, in order to investigate correlations between the activated hidden units.
- Determine the effectiveness of various sets of parameters of the RBM (varying also the type of unit and states implemented) to find which ones have a bigger impact on its generative power.

This review is organized as follows. Section 2 presents the key concepts and algorithms underlying the structure and training of the implemented models. Section 3 outlines the main steps of the models' actual implementation. Section 4 reports the obtained results. Finally, in Section 5 the conclusions drawn from the analysis are discussed.

2. METHODS

Dataset – The MNIST 784 dataset, retrieved from OpenML [4], consists of a set of handwritten digits with 784 binary pixels, centered in a 28×28 black and white image. For the purpose of this study, a subset containing the digits '7', '8' and '9' is extracted, resulting in a total of 21076 samples. Figure 1 shows an example from the dataset.

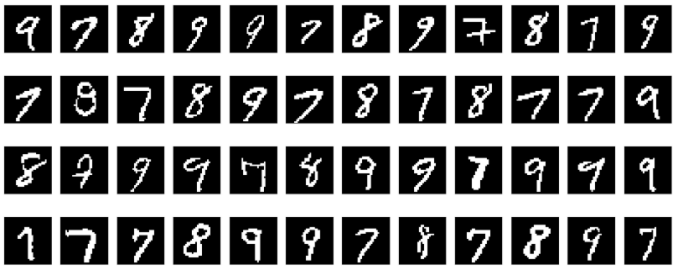


FIG. 1: Example of MNIST selected binarized digits.

Training – In training RBMs, it is possible to choose between different configurations regarding the structure of units and states:

- **Bernoulli units**, discrete and binary.
- **Ising units**, from the Ising model, $\{-1, +1\}$, later referred to as `SPINS`.
- **One-hot states**, only possible with Bernoulli units and later referred to as `POTTS`.

The configuration of the visible and hidden units has an energy given by the following function:

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

where $a_i(\cdot)$ and $b_\mu(\cdot)$ are visible and hidden biases, and $w_{i\mu}$ is the weight between the variables.

The probability of every possible pair of a visible and hidden vector is given by:

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

where the partition function Z is given by summing over all the possible pairs:

$$Z = \sum_{v,h} e^{-E(v,h)}$$

Finally, the probability assigned to a visible vector is the sum over all the possible hidden vectors:

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

Given the bipartite structure of RBMs, their visible and hidden units are conditionally independent. For this reason, the probabilities of activating specific hidden and visible units are given by:

$$p(h_\mu = 1|v) = \sigma(b_\mu + \sum_i v_i w_{i\mu}) \tag{2}$$

$$p(v_i = 1|h) = \sigma(a_i + \sum_\mu h_\mu w_{i\mu}) \tag{3}$$

where $\sigma(x) = \frac{1}{1+e^{-x}}$ is the sigmoid function.

The minimization of the cost function can be approached in different ways. Many generative models follow the *Maximum Likelihood Estimation* (MLE) method, which seeks parameters $\hat{\theta}$ that maximize the likelihood of the training set. Usually in this case, the negative log-likelihood is chosen as the cost function. However, approximating the gradient of this distribution with respect to its parameters is computationally expensive. In alternative, sampling-based methods and approximate gradient methods such as Gibbs Sampling (GS) and Contrastive Divergence (CD) can be implemented.

Log-likelihood − When using the Bayesian approach, the prior distribution and the likelihood function are combined to obtain the posterior distribution

$$p(\theta|V) = \frac{p(V|\theta)\, p(\theta)}{\int d\theta' p(V|\theta')\, p(\theta')}$$

which describes the knowledge about parameters θ after observing the data V. Often it is not possible to analytically compute the denominator of this distribution, hence Markov Chain Monte Carlo methods are required to draw random samples of $p(\theta|V)$. The goal is to identify the parameter $\hat{\theta}$ that maximizes the likelihood, or equivalently the log-likelihood since the logarithmic function is monotonic

$$\hat{\theta} = \arg_\theta \max \log p(V|\theta).$$

The log-likelihood per data point $\ell_\theta(v)$ is given by [2]:

$$\ell_\theta(v) = \ln \sum_h e^{-E(v,h)} - \ln \sum_{v'} \sum_h e^{-E(v',h)}$$

where the second term is the partition function Z. The average over M data points gives the overall log-likelihood

$$\mathcal{L} = \frac{1}{M} \sum_{m \le M} \ell_\theta\left(v^{(m)}\right).$$

The computation of the partition function Z is intractable. The hard part lies in summing up the Boltzmann weights of all possible configurations in Z: with D visible units and L hidden units there are 2^{D+L} possible configurations. An alternative procedure is suggested by Baiesi [1]. By defining

$$H_i(h) = a_i + \sum_\mu w_{i\mu} h_\mu$$

the Boltzmann weight takes the form:

$$e^{-E(v,h)} = \prod_\mu e^{b_\mu h_\mu} \prod_i e^{H_i(h) v_i} \tag{4}$$

In Equation 4 the first factor is the contribution of hidden units to the energy, defined as $G(h)$. This leads to a reduced partition function $Z(h)$ defined as

$$Z(h) = G(h) \prod_i \left(1 + e^{H_i(h)}\right)$$

This means computing

$$\ln Z = \ln \left[\sum_h G(h) \prod_{i=1}^{D} \left(1 + e^{H_i(h)}\right) \right]$$

for Bernoulli units, while using Ising units:

$$\ln Z = \ln \left[\sum_h G(h) \prod_{i=1}^{D} \left(1 + \cosh\left(H_i(h)\right)\right) \right].$$

This is easier to compute and it becomes numerically stable when limiting the argument to avoid overflowing. To obtain $\mathcal{L}$, it is sufficient to average over $v^{(m)}$ points of the dataset.

Gibbs Sampling and Contrastive Divergence – In the case of binary units, the probability given by Equation 1 leads to the learning rule (given a learning rate ϵ) [3]:

$$\Delta w_{i\mu} = \epsilon(\langle v_i h_\mu \rangle_{data} - \langle v_i h_\mu \rangle_{model}). \tag{5}$$

The expectation values with respect to the data are given by averaging all samples in a minibatch where the visible units are set in agreement with the observed values, and then using Equation 2 for the hidden units. The expectation values with respect to the model are more difficult to find and require an iterative technique called Gibbs sampling (GS). One iteration of GS, given the current state t of the variables, samples sequentially from the conditional distributions $h_{t+1} \sim p(h|v_t)$ and $v_{t+1} \sim p(v|h_t)$ to update the state of each variable. At $t \to \infty$, the samples will converge to the true model distribution.

Contrastive Divergence (CD) is an approximation of this sampling technique. It consists in limiting the number of iterations that are performed in order to make the model easier to train. Even if RBMs typically learn better models if more steps are performed [3], the approximation is already reasonably good after only one iteration. Using CD, however, means sacrificing some ability to generalize, since the samples after only a few steps of GS will be more distant from the true model distribution.

Initialization of biases – The initialization proposed by Hinton [3] for the bias of a visible unit is applied by using the formula:

$$a_i = \log\left(\frac{p_i}{1-p_i}\right) \tag{6}$$

where p_i is the fraction of samples in which the i-th unit is active. When updating the visible states, this value corresponds to the probability that a visible unit is equal to 1, and it is used to stochastically sample the state of each visible unit. The activation probability of a visible neuron is given by Equation 3. By setting $w_{i\mu} \sim 0$, it follows that $P(v_i = 1) \sim \sigma(a_i)$. This ensures that the neuron's average activation matches the empirical distribution of the data [2]. This helps stabilize the learning process and prevents the model from converging too slowly.

Optimizers – Once the contrastive divergence gradient is estimated, the parameters are updated using gradient-based optimization methods. The implemented optimizers are SGD, RMSprop and Adam. SGD and RMSprop are both implemented with and without momentum, following the pseudocodes available in PyTorch.

Validation – The log-likelihood score obtained for each model is validated by setting N different random seeds and repeating the training N-times, then computing the mean of the log-likelihood score and standard deviation. The goal of this procedure is to account for the stochastic nature of the training process, quantifying its effect on the performance of each model.

3. IMPLEMENTATION

Random Search – We implement a random search to tune the machine hyperparameters as shown in Table I. We select a few models by comparing their learning curves, and cross-validate their log-likelihood at the appropriate epoch (see Table II and Figure 2).

$\hat{\theta}$	Searched Interval	Description
`L`	$[3, 10]$	Hidden units
`SPINS`	*boolean*	Ising units
`POTTS`	*boolean*	One-hot encoding
`Gamma`	$[10^{-4}, 0]$	Regularization
`Grad`	SGD, RMSprop, Adam	Optimization function
`Nt`	$[1, 15]$	CD steps
`Nepoch`	$[50, 350]$	Epochs
`Nmini`	$[10, 50]$	Minibatches per epoch
`Nini`	$[5, 50]$	Initial minibatch's size
`Nfin`	$[$`Nini`$, 350]$	Final minibatch's size

TABLE I: Selected hyperparameters $\hat{\theta}$ in the random search and respective interval of search.

Mod.	`L`	`Gamma`	`Nt`	`Npochs`	`Nmini`	`Nini`	`Nfin`	`Grad`
0	7	0.001	4	300	30	5	100	RMSprop
1	10	0.001	1	200	10	5	325	RMSprop
2	5	0.001	2	190	45	10	450	SGD
3	10	0.01	1	200	10	5	10	RMSprop
4	10	0.01	1	200	10	5	10	RMSprop
5	10	0.001	7	350	10	5	325	RMSprop

TABLE II: The "Mod." column denotes the model labels, the subsequent parameters are described in Table I. Note that `POTTS` and `SPINS` are not shown as the only acceptable model with one-hot encoding resulted to be model 3, while no `SPINS` model displayed satisfactory scores.

From the search, we derive some useful information:

1. It is preferable to use a high number of hidden units, and to implement the RMSprop optimizer.

2. One-hot encoding and the Ising units architecture does not seem to work.

We tune the optimizers' parameters to check the validity of the first point. It turns out that no optimizer is actually better than the others if the parameters are set correctly. One could then choose the optimizer with less parameters to tune or the most efficient in terms of computational time[1].

[1] With the exception of one-hot encoding architectures as discussed in Section 4.

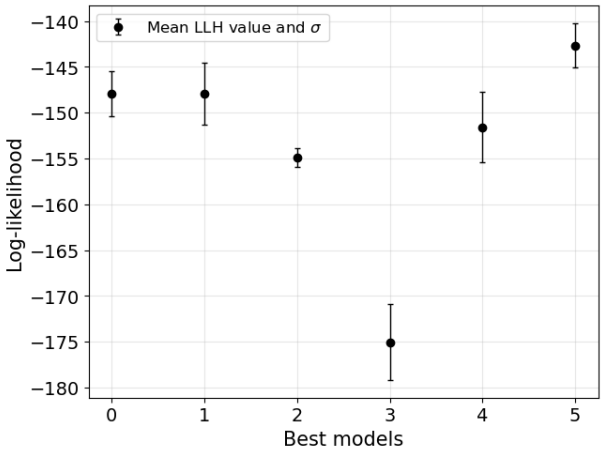


FIG. 2: Scores of the models selected from the random search. On the x axis the model labels are indicated, their architecture is described in Table II.

Concerning the second point, while with one-hot encoding we obtain bad, but somewhat acceptable results (model 3 has mean $\mathcal{L} \sim -175$ as shown in Figure 2), the Ising model scores reach at most ~ -560. We try to investigate this phenomenon by fine-tuning the hyperparameters and the optimizers' parameters. We could not raise the Ising scores, but — thanks to the lighter computation of one-hot encoding — it proved to be easier to build better models of this kind, reaching $\mathcal{L} \sim -145$.

The main issue with building a better RBM with no one-hot encoding, is the computational cost of calculating $\mathcal{L}$, as the number of possible configurations of hidden units for this type of model is 2^L. We choose as number of hidden units for our model $L = 8$, since higher numbers lead to excessive computational time or system failure.

$\mathcal{L}$ and CD-steps – To understand if there is a correlation between the number of CD steps and the behaviour of $\mathcal{L}$, we trained various models trying different values of the former `Nt` $\in [2, 4, 6, 8, 10, 12]$. We concluded that the number of CD steps does not significantly affect the log-likelihood, consequently, we adopted a low number of steps.

Generating Numbers – Eventually, we test the generative power of the model. This is accomplished by iteratively sampling from the conditional distributions of hidden and visible units in the same way it is performed for the CD procedure, but without updating weights and biases. Furthermore it is possible to multiply the hidden units' weights by a factor bigger than 1, which corresponds to lowering the temperature in a Boltzmann distribution. This procedure reduces the noise at the cost of reducing the variability of the generated numbers as well.

4. RESULTS

Best Models – The two best models derived from Section 3 are shown in Table III. These are referred to as L8 and L55, from the corresponding number of hidden units. A third model (L5) is displayed as a comparison with machines composed of five hidden units. Their learning curve is presented in Figure 3 and the cross-validated scores are shown in Figure 4.

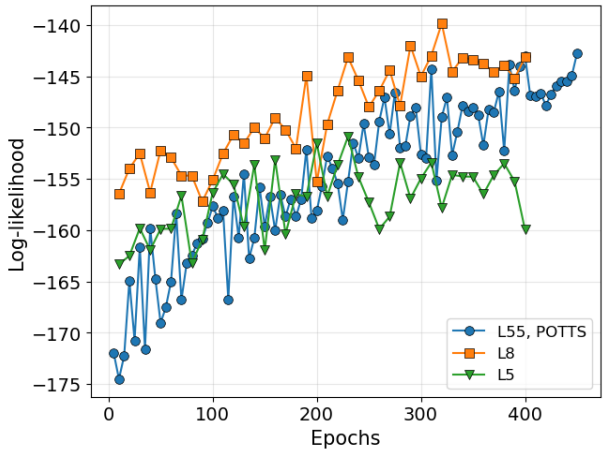


FIG. 3: Learning curve of the three best models. $\mathcal{L}$ is computed every 5 epochs for L55 model, and every 10 epochs for L5 and L8.

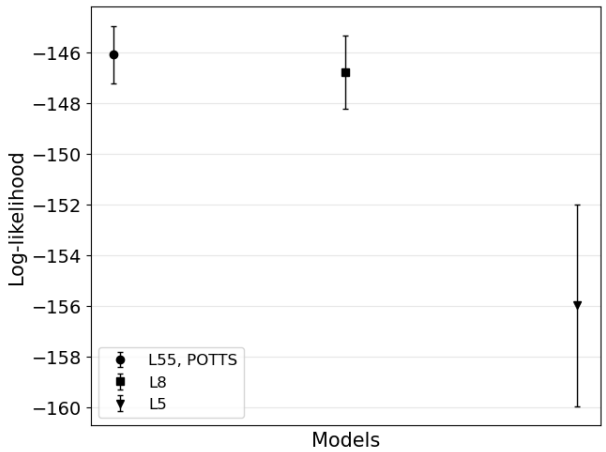


FIG. 4: Cross validated scores of the best models.

The tweaking of L8 does not bring an increase of the scores. On the other hand, the tuning of L55 — which uses one-hot encoding — proves to be much less straightforward. The scores do not increase with the number of hidden units with respect to non-`POTTS` models, as L55 activates only a fraction of them. It is only with the introduction of momentum that it manages to break this behaviour and get results as good as the L8 model, if not

better.

We set an initial momentum value to 0.2 and, when the training reaches 60% of the process, we increase it to 0.5. The learning rate is initially set at 0.1, and then it linearly decreases to 0.01.

Figure 5 shows the plots of the weights and biases of the hidden units at the final epoch of the training, where it clearly appears that only three units are not activated.

The number of activated units also appear to decrease when the number of hidden units is too high: with 100 hidden units only 33 activate.

$\hat{\theta}$	L5	L8	L55
`L`	5	8	55
`SPINS`	False	False	False
`POTTS`	False	False	True
`Gamma`	0.001	0.001	0.1
`Grad`	RMSprop	RMSprop	RMSprop
`Nt`	7	2	1
`Nepoch`	400	400	450
`Nmini`	100	20	50
`Nini` $\rightarrow$ `Nfin`	$3 \rightarrow 20$	$10 \rightarrow 500$	$10 \rightarrow 300$
`m` $\rightarrow$ `M`	0	0	$0.2 \rightarrow 0.5$
`L. rate`	0.05	0.05	$0.1 \rightarrow 0.01$

TABLE III: Hyperparameters of the three best models. `m` $\rightarrow$ `M` are the starting and final momentum, respectively. `L. rate` is the learning rate.

Hidden space activation – We analyze the hidden space activations of the three models introduced above (see Table III).

Since it is more readable than the others, only L5 data is shown in Figure 6b. This reveals several patterns: digits labeled as '8' consistently show the least diverse unit activation, followed by '9', while digits labeled as '7' typically activate most of the available units.

Generative Power – Figure 7 shows a test of the generative power of the three models, where new images of the given digits are produced.

L5 and L8 sometimes fail to draw the correct numbers (generating '9' instead of '7' and '8').

5. CONCLUSIONS

We built an RBM designed to learn and generate new images of the digits '7', '8' and '9', given a sample of binarized images extracted from the MNIST 784 database. Our implementation included Random Search, Contrastive Divergence, and some optimization techniques, using log-likelihood as the score.

From the analysis of the hidden space activation we better understood the process followed by the model to generate new digits. Since the bias (Figure 6a) already represents the digit '8', most of the time the machine does not need to activate any hidden unit to generate it. This is evident in the plot in Figure 6b, which shows the frequency of activation of each unit and state: state number 0, which corresponds to no activated hidden units, has more than $60,000$ counts. Then, to generate the other digits, it usually uses a combination of hidden unit 0 (which represents the negative of the bias) and the others.

We analyzed the behavior of $\mathcal{L}$ and got the best improvement by increasing the number of hidden units. This is in agreement with what Hinton suggests [3]: it would be optimal to have the same number of hidden units as the length of the data vector, meaning ~ 800 hidden units. The number of CD steps mattered little on the increase of the final score, with improvements of a couple of points.

We selected two best models, one has a high number of hidden units (L55), and also implements one-hot encoding for their activation, and one with a lower number (L8). The respective log-likelihood are:

$$\mathcal{L}_{L55} = -146 \pm 1$$
$$\mathcal{L}_{L8} = -147 \pm 1$$

Even though they have almost identical scores, we considered L55 to be superior in terms of generative power to L8 as the latter struggles with digits '7' and '8', by sometimes outputting a '9'. One of the problematic features is the horizontal stroke of the number '7', which – being an optional detail – does not consistently appear in the training set. This issue can also be appreciated by considering the generated digits of L55 (Figure 7): this model is always able to produce the correct number, but it would sometimes activate only a few pixels of the horizontal bar, leading to the loss of the feature if the weights are multiplied.

Given access to more capable computational resources, it might be possible that the standard RBM configuration could outperform the `POTTS` variant. However, due to hardware limitations, we were unable to verify this hypothesis.

CONTRIBUTIONS

Bellavia and Belli worked on the activation of the hidden states analysis, and the implementation of the log-likelihood computation.

Maida and Lera focused on the implementation of new optimizers and the evaluation of the generative power.

All the authors contributed in running and tuning various models of the RBM, as well as writing and revising the final document.

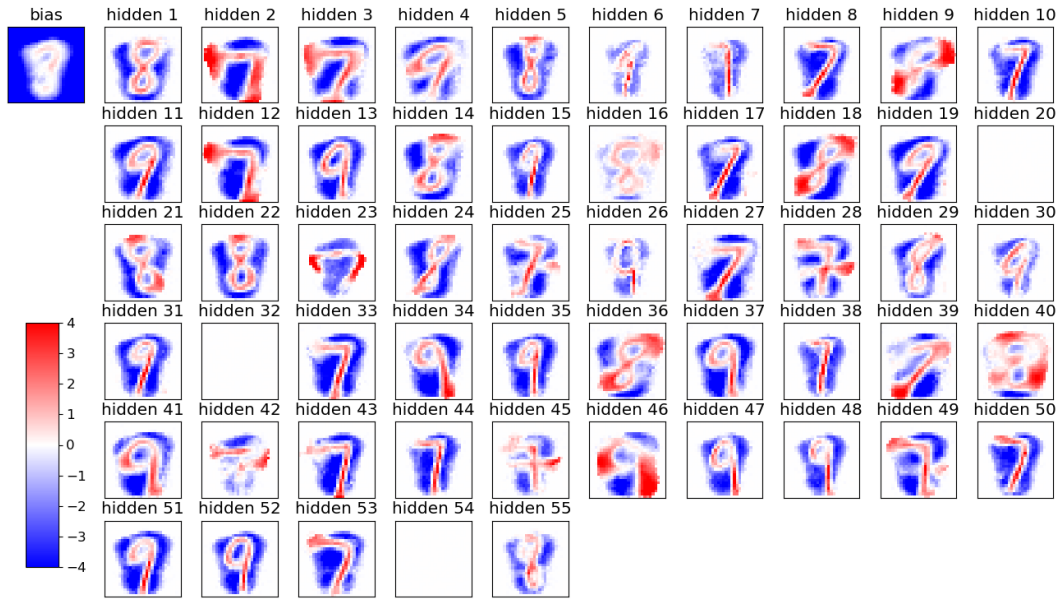


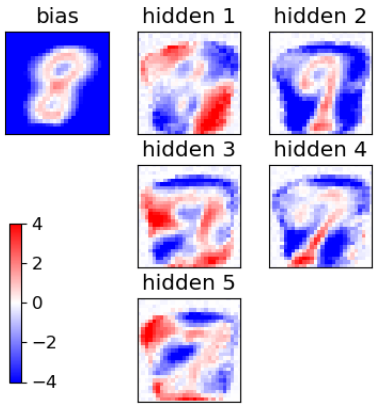
FIG. 5: Bias and weights plots of hidden units at the final epoch of L55 training.

[1] Marco Baiesi **Log-likelihood computation for restricted Boltzmann machines**, 1–2 (2025).

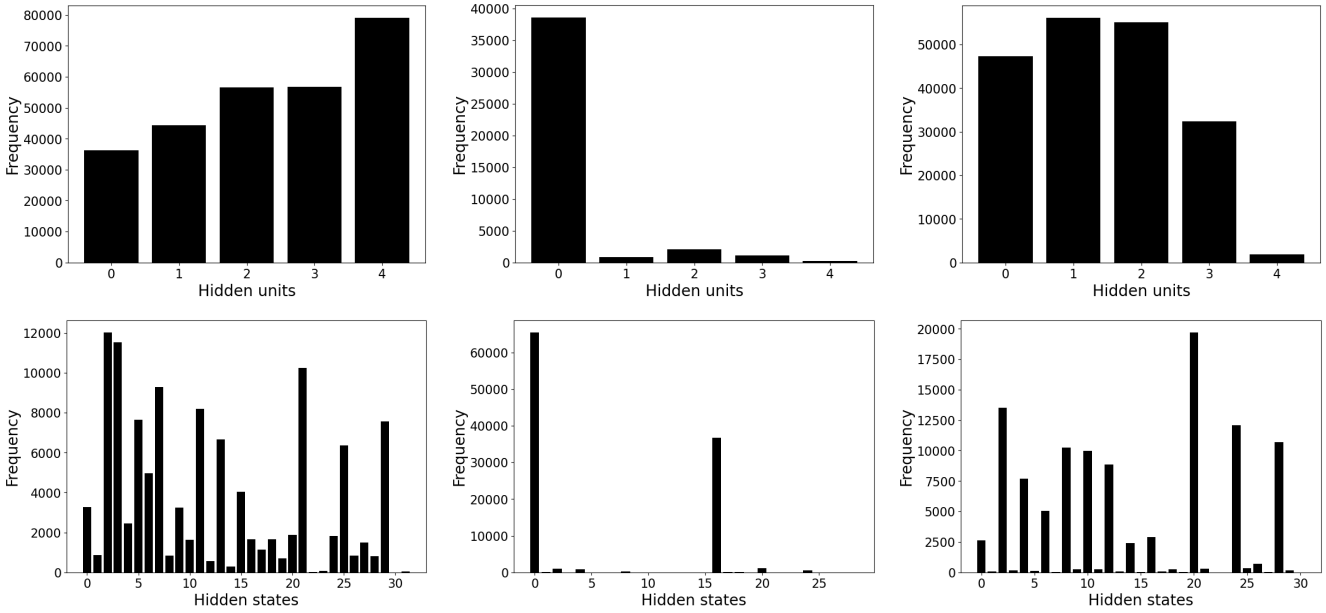
[2] P. Mehta, M. Bukov, et al. **A high-bias, low-variance introduction to Machine Learning for physicists**, 21 (2018).

[3] G. Hinton, **A Practical Guide to Training Restricted Boltzmann Machines**, (2012)

[4] M. Feurer, et al. **OpenML-Python: an extensible Python API for OpenML**, (2019)

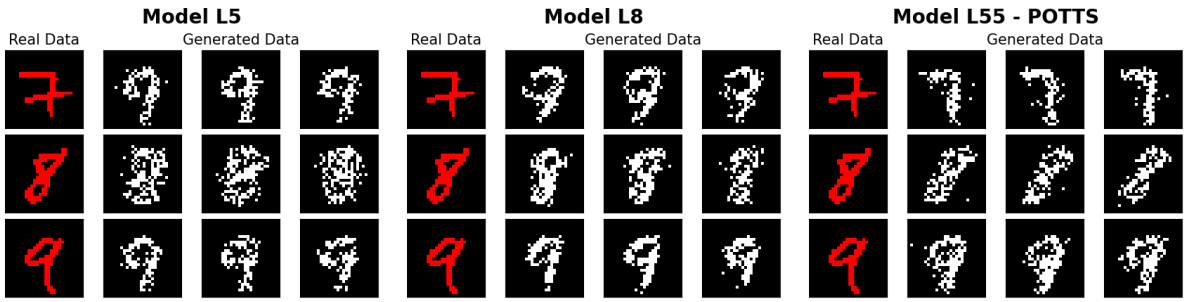


(a) Bias and weights of final epoch, L5

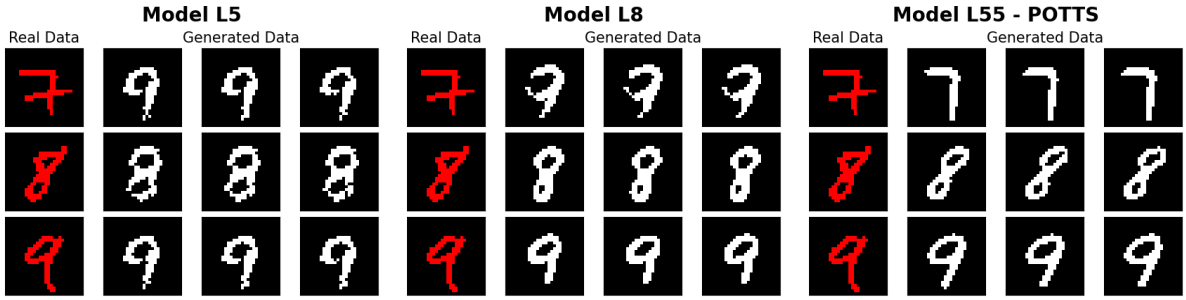


(b) In these panels are shown hidden units activations and hidden state activations for each digit, '7' (left column), '8' (center column) and '9' (right column), model L5.

FIG. 6



(a) No multiplication applied.



(b) Multiplication of 20.

FIG. 7: First three numbers generated by the models along with the provided data (in red).