# Assignment-02
## Restricted Boltzmann Machine (RBM)

**Jagabandhu Mishra 183081002**
**Seema R. K 173021001**
**Sawan Singh Mahara 191081003**
**Akshay A. J 191081001**

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 Evolution: Feedback Neural Network to Restricted Boltzmann Machine

Neural networks can be used for various tasks right from pattern storage to pattern classification with appropriate selection of weights and network topology/architecture.

### 1.1.1 Feedback Neural Network and its limitations

Feedback neural network (FBNN) are a class of neural networks where the output of each processing unit/neuron acts as input to all other processing units and to itself. The amount of information that acts as input to the units is weighted by the strength of the link between the units. By choosing appropriate parameters for this network it can be used to perform tasks like auto-association.

The main goal of auto-associative networks is to learn the pattern associated with input in training phase and recall the pattern when erroneous or noisy version of input is fed to the network during the testing phase i.e. it should possess accretive behavior. If the processing units in the FBNN are linear this behavior cannot be achieved as an erroneous input will lead to the generation of erroneous output instead of pattern recall. This drawback can be overcome by replacing the linear processing units with non-linear processing units. Hence, the FBNN with non-linear processing units can be used for the task of pattern storage.

The task of pattern storage involves learning the features and spatial relationship between the features associated with the input with the aim of recollecting the exact input pattern when

an approximate version of the pattern is provided as input to the network. If we consider the output (depends on weights and biases) of each unit as energy of that unit/state, the path that state traverses in the energy landscape (energy as a function of state) at successive instants of time can be termed as the trajectory of that sate. The trajectory is determined by the activation dynamics model used for the network.

When an input is applied to the network, the activation dynamics is applied to input continuously till it reaches an equilibrium state. Theses equilibrium states are the minimum energy points explored to store the patterns in the network during the pattern storage task. It is the presence of these energy minima's that help in pattern retrieval when an approximate version of input is fed to the network.

The two parameters that need to be evaluated in FBNN are error in recall and the capacity of the network. The error in recall occurs when the network recollects a wrong pattern when fed with noisy input. This can reduced by adjusting the weights of the network to ensure that the energy landscape of the network matches the probability distribution of the pattern to be stored. Capacity of the network is defined as the number of patterns a network can store. It depends on the number of processing units available in the network. If the network has N binary processing units, though it can have $2^N$ states, the network can store only N binary patterns as the network will have only N minimum energy states. Hence, the capacity of the network is determined by the number of processing units and their interconnections in the network.

If the number of patterns to be stored are more than the number of energy minima's it is a hard problem and if the number of minima's are more than the number of patterns to be stored there will be increase in false wells which in turn leads to increase in error in recall.

### 1.1.2 Hopfield Model

Hopfield model is a fully connected (FC) FBNN with symmetric weights and no self-loop. The weights are updated asynchronously and the energy associated with each of the state reduces or remains same as the state of the network changes. The number of patterns to be stored is limited and is of the order of number of units.

Based on the rule used to update the states and output function of each unit, the Hopfiled Model can be continuous or discrete. Discrete Hopfield model the state update is synchronous and the output function can be binary/bipolar whereas in a continuous Hopfield model the states are updated by activation dynamics and output function is continuous and non-linear.

### 1.1.3 Boltzmann machine and Restricted Boltzmann machine

In Boltzmann machine, the errors in recall due to false minima is resolved by updating the states stochastically and asynchronously. Also, the probability distribution of patterns are considered

while designing the energy minima so that the given patterns correspond to the lowest energy minima in the network.
The hard problem in the pattern storage task is handled by introducing additional units in the feedback network called hidden units. The units where input is applied are called visible units.

Though the Boltzmann machine appear to solve the pattern storage problem they are slow and difficult to train. Hence, Restricted Boltzmann machine was introduced. Restricted Boltzmann machines do not have connection in between the units of the same layer. Hence, easier to train as compared to Boltzmann machine.

# Chapter 2

# Restricted Boltzmann Machine (RBM)

## 2.1 RBM

Restricted Boltzmann machine (RBM) is the restricted version of stochastic neural network called Boltzmann machine. Boltzmann machine is a feedback network with hidden units that supports stochastic updates (refer Fig. 2.1). Feedback networks are designed to solve pattern storage problems, later it extends to solve other pattern recognition tasks. Feedback networks were proposed to mimic the characteristics of biological neurons. Unfortunately, due to many computational and analytical limitations it ended with Boltzmann machines [1], after observing that Boltzmann machines need a large amount of training data and computational resources. Parallely, different types of statistical modelling techniques evolved that provide far better performance using comparatively less resources than the Boltzmann machines ever did. One such solution proposed that RBM should run by removing the interconnections between the visible units and the hidden units(refer Fig. 2.2).

From the feedback network, we can write the equation of energy corresponding to a given state as:

$$E(V, H) = -\sum_{i}\sum_{j} w_{ij} v_i h_j - \sum_{i} b_i v_i - \sum_{j} c_j h_j \tag{2.1}$$

and the probability of a visible and hidden unit state can be given as:

$$P(V, H) = \frac{e^{-E(V,H)}}{\sum_V \sum_H e^{-E(V,H)}} \tag{2.2}$$

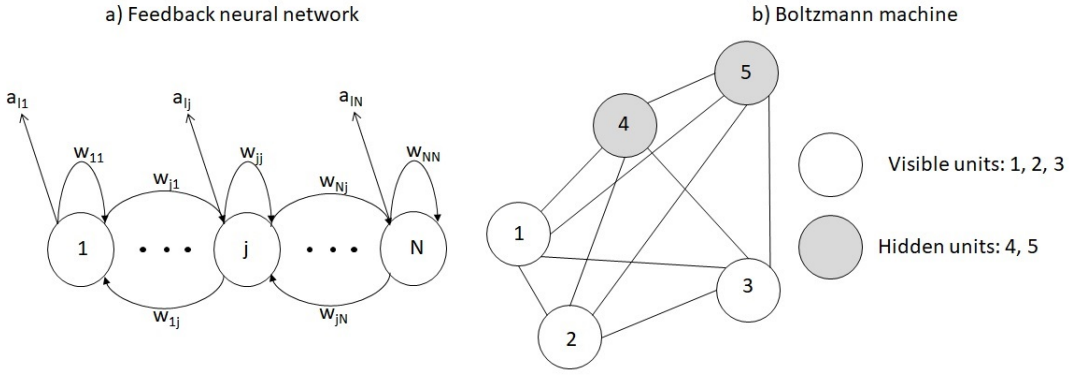


FIGURE 2.1: Network architectures a) Feedback neural network b) Boltzmann machine

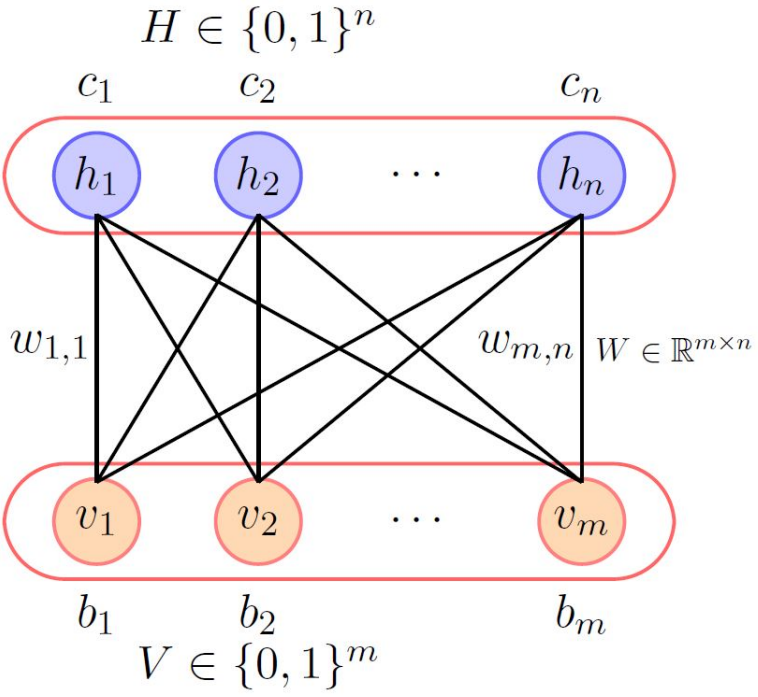


FIGURE 2.2: RBM

where $V \in \{0,1\}^m$, $H \in \{0,1\}^n$ for a binary RBM or $V \in R^m$, $H \in R^n$ for a continuous RBM , m and n is the number of visible and number of hidden units respectively.

## 2.2 RBM Training

The training in RBM is unsupervised. Given the data points X as input to the visible units, we have to learn $P(V,H)$ which is parameterized as $P(V,H) = \frac{e^{-E(V,H)}}{\sum_V \sum_H e^{-E(V,H)}}$, where the hidden units are unknown.

As we want to store the pattern $X \in V$ in the network, we have to capture $P(X)$, which can be written as $\sum_H P(V,H)$.

So, for a given dataset D ($D = \{x_1, x_2, ..., x_l\}$), we have to maximize $\prod_{i=1}^{l} P(x_i|\theta)$, where $\theta$ are the parameters $(w_{ij}, c_i, b_j)$ of the neural network.

The objective function can be written as

$$\begin{aligned} lnL(\theta) =& ln \prod_{i=1}^{l} P(x_i|\theta) \\ =& \sum_{i=1}^{l} lnP(x_i|\theta) \\ =& \sum_{i=1}^{l} lnP(V_i|\theta) \\ =& \sum_{i=1}^{l} ln \sum_{H} \frac{e^{-E(V,H)}}{\sum_V \sum_H e^{-E(V,H)}} \end{aligned} \tag{2.3}$$

we have to maximize $\ln(L(\theta))$ i.e $\max_\theta \sum_{i=1}^{l} \ln \sum_H \frac{e^{-E(V,H)}}{\sum_V \sum_H e^{-E(V,H)}}$.

We can use stochastic gradient approach to update the parameters of the network, which will maximize the objective function, for that we have to compute $\frac{\partial lnL(\theta)}{\partial \theta}$.

$$\begin{aligned} \frac{\partial lnL(\theta)}{\partial \theta} =& \frac{\partial}{\partial \theta} \left[ ln \sum_H e^{-E(V,H)} - ln \sum_{V,H} e^{-E(V,H)} \right] \\ =& -\sum_H \frac{e^{-E(V,H)}}{\sum_H e^{-E(V,H)}} \frac{\partial E(V,H)}{\partial \theta} + \sum_{V,H} \frac{e^{-E(V,H)}}{\sum_{V,H} e^{-E(V,H)}} \frac{\partial E(V,H)}{\partial \theta} \\ =& -\sum_H P(H|V) \frac{\partial E(V,H)}{\partial \theta} + \sum_{V,H} P(V,H) \frac{\partial E(V,H)}{\partial \theta} \end{aligned} \tag{2.4}$$

Now our $\theta = \{w_{ij}, c_i, b_j\}$, so we have to compute $\frac{\partial lnL(\theta)}{\partial w_{ij}}$, $\frac{\partial lnL(\theta)}{\partial b_j}$ and $\frac{\partial lnL(\theta)}{\partial c_i}$ one by one.

$$\begin{aligned} \frac{\partial lnL(\theta)}{\partial w_{ij}} =& \sum_H P(H|V) h_i v_j - \sum_{V,H} P(V,H) h_i v_j \\ =& \sum_H P(H|V) h_i v_j - \sum_V P(V) \sum_H P(H|V) h_i v_j \end{aligned} \tag{2.5}$$

From eq. 2.5 it has been seen that there is common term $\sum_H P(H|V) h_i v_j$ need to be solved.

$$\begin{aligned} \sum_H P(H|V) h_i v_j =& \sum_{H_i} P(H_i|V) P(H_{-i}|V) h_i v_j \\ =& \sum_{H_i} P(H_i|V) h_i v_j \sum_{H_{-i}} P(H_{-i}|V) \\ =& P(H_i = 1|V) v_j \end{aligned} \tag{2.6}$$

The $P(H_i = 1|V)$ can be written as:

$$\begin{aligned} P(H_i = 1|V) =& P(H_i = 1|H_{-i}, V) \\ =& \frac{e^{-E(H_i=1,H_{-i},V)}}{e^{-E(H_i=1,H_{-i},V)} + e^{-E(H_i=0,H_{-i},V)}} \\ =& \frac{1}{1 + e^{-(-\sum_l w_{li} v_l - c_i)}} \\ =& \sigma(\sum_l w_{li} v_l + c_i) \end{aligned} \tag{2.7}$$

Now, $\frac{\partial \ln L(\theta)}{\partial w_{ij}}$ can be written as:

$$\begin{aligned} \frac{\partial \ln L(\theta)}{\partial w_{ij}} =& \sigma(\sum_j w_{ij} v_j + c_i) v_j - \sum_V P(V) \sigma(\sum_j w_{ij} v_j + c_i) v_j \\ \nabla_w L(\theta) =& \sigma(WV + C)V^T - \sum_V P(V) \sigma(WV + C)V^T \\ \nabla_w L(\theta) =& \sigma(WV + C)V^T - E_V[\sigma(WV + C)V^T] \end{aligned} \tag{2.8}$$

Similarly, $\nabla_b L(\theta)$, $\nabla_c L(\theta)$ can be written as,

$$\begin{aligned} \nabla_b L(\theta) =& V - E_v[v] \\ \nabla_c L(\theta) =& \sigma(WV + C) - E_V[\sigma(WV + C)] \end{aligned} \tag{2.9}$$

From the equations of $\nabla_w L(\theta)$, $\nabla_b L(\theta)$ and $\nabla_c L(\theta)$, there are three expectation terms in all the cases, which require us to compute the expectation over all the possible states of all visible units. If we consider V to take only binary values; the possible number of state sequence will be $2^m$ which increases exponentially with increase in the number of visible unit m. In general, to compute the expectation instead of using all population (possible states) sample of states can be used. The samples are drawn using **Gibbs Sampling** (uses Markov chain property to learn the stationary distribution). Suppose k is number of steps require to get the stationary distribution, then generate r number of samples from the stationary distribution and take the average of that to compute the expectation terms. Hence the expectation terms can be rewritten as:

$$\begin{aligned} E_V[\sigma(WV + C)V^T] =& \frac{1}{r} \sum_{i=k+1}^{k+r} \sigma(WV^{(i)} + C)V^{(i)T} \\ E_V[V] =& \frac{1}{r} \sum_{i=k+1}^{k+r} V^{(i)} \\ E_V[\sigma(WV + C)] =& \frac{1}{r} \sum_{i=k+1}^{k+r} \sigma(WV^{(i)} + C) \end{aligned} \tag{2.10}$$

Finally the update equation can be written as using Gibbs sampling as:

$$\begin{aligned} \mathrm{W} =& \mathrm{W} + \sigma(\mathrm{WV_d} + \mathrm{C})\mathrm{V_d^T} - \frac{1}{\mathrm{r}} \sum_{\mathrm{i=k+1}}^{\mathrm{k+r}} \sigma(\mathrm{WV^{(i)}} + \mathrm{C})\mathrm{V^{(i)T}} \\ \mathrm{b} =& \mathrm{b} + \mathrm{V_d} - \frac{1}{\mathrm{r}} \sum_{\mathrm{i=k+1}}^{\mathrm{k+r}} \mathrm{V^{(i)}} \\ \mathrm{C} =& \mathrm{C} + \sigma(\mathrm{WV_d} + \mathrm{C}) - \frac{1}{\mathrm{r}} \sum_{\mathrm{i=k+1}}^{\mathrm{k+r}} \sigma(\mathrm{WV^{(i)}} + \mathrm{C}) \end{aligned} \tag{2.11}$$

where W,b,C, $\mathrm{V_d}$ and V are the weight, bias visible unit, bias hidden unit, training input data (to visible unit) sampled state (corresponding to visible unit) respectively. The full algorithm of RBM training using Gibbs sampling is depicted in Fig. 2.3.

**Algorithm 0:** RBM Training with Block Gibbs Sampling

**Input:** RBM $(V_1, ..., V_m, H_1, ..., H_n)$, training batch $D$
**Output:** Learned Parameters $\mathbf{W}, \boldsymbol{b}, \boldsymbol{c}$
init $\mathbf{W}, \boldsymbol{b}, \boldsymbol{c}$
**forall** $\boldsymbol{v} \in D$ **do**
  Randomly initialize $\boldsymbol{v}^{(0)}$
  **for** $t = 0, ..., k, k+1, ..., k+r$ **do**
    **for** $i = 1, ..., n$ **do**
      sample $h_i^{(t)} \sim p(h_i|\boldsymbol{v}^{(t)})$
    **end**
    **for** $j = 1, ..., m$ **do**
      sample $v_j^{(t+1)} \sim p(v_j|\boldsymbol{h}^{(t)})$
    **end**
  **end**
  $\mathbf{W} \leftarrow \mathbf{W} + \eta[\sigma(\mathbf{W}\boldsymbol{v}_d + \boldsymbol{c})\boldsymbol{v}_d^T - \frac{1}{r}\sum_{t=k+1}^{k+r} \sigma(\mathbf{W}\boldsymbol{v}^{(t)} + \boldsymbol{c})\boldsymbol{v}^{(t)T}]$
  $\boldsymbol{b} \leftarrow \boldsymbol{b} + \eta[\boldsymbol{v}_d - \frac{1}{r}\sum_{t=k+1}^{k+r} \boldsymbol{v}^{(t)}]$
  $\boldsymbol{c} \leftarrow \boldsymbol{c} + \eta[\sigma(\mathbf{W}\boldsymbol{v}_d + \boldsymbol{c}) - \frac{1}{r}\sum_{t=k+1}^{k+r} \sigma(\mathbf{W}\boldsymbol{v}^{(t)} + \boldsymbol{c})]$
**end**

FIGURE 2.3: RBM Algorithm

The drawback of the Gibbs sampling based approach is that, it could take many iterations (k) to reach the stationary distribution since we initialize $\mathrm{V^0}$ randomly. There is an approach called k-contrastive divergence approach, where the $\mathrm{V^0}$ is initialized to the current training data sample $\mathrm{V_d}$ following which, the vanilla Gibbs Sampling is run. for k steps. Let the sample at the $\mathrm{k^{th}}$ step be denoted by $\tilde{\mathrm{V}}$(refer Fig. 2.4).

The update rule for k-contrastive divergence approach can be written as per eq. 2.12:

$$\begin{aligned} \mathrm{W} =& \mathrm{W} + \sigma(\mathrm{WV_d} + \mathrm{C})\mathrm{V_d^T} - \sigma(\mathrm{W\tilde{V}} + \mathrm{C})\tilde{\mathrm{V}}^{\mathrm{T}} \\ \mathrm{b} =& \mathrm{b} + \mathrm{V_d} - \tilde{\mathrm{V}} \\ \mathrm{C} =& \mathrm{C} + \sigma(\mathrm{WV_d} + \mathrm{C}) - \sigma(\mathrm{W\tilde{V}} + \mathrm{C}) \end{aligned} \tag{2.12}$$

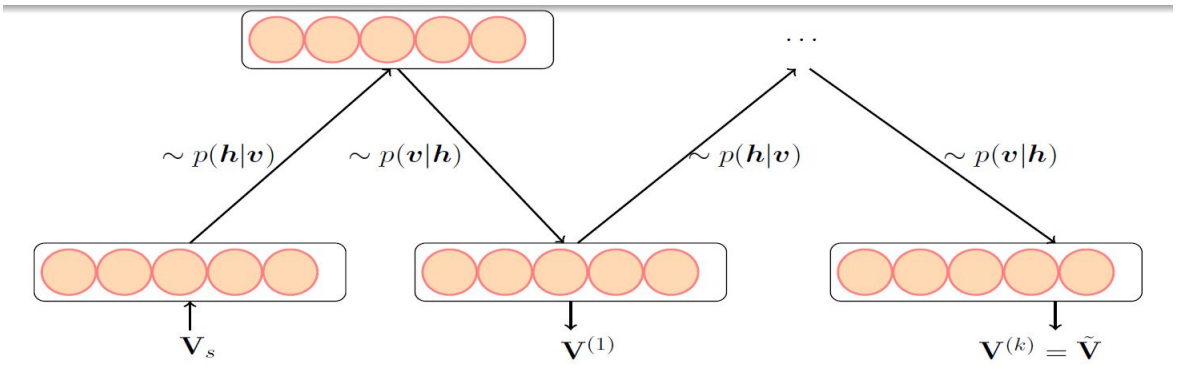


FIGURE 2.4: Contrastive divergence

In practice k = 1 is work fine. The detailed algorithm to train RBM using k-contrastive divergence approach is depicted in Fig. 2.5.

**Algorithm 0:** k-step Contrastive Divergence

**Input:** RBM $(V_1, ..., V_m, H_1, ..., H_n)$, training batch $D$
**Output:** Learned Parameters $\mathbf{W}, \boldsymbol{b}, \boldsymbol{c}$
init $\mathbf{W} = \boldsymbol{b} = \boldsymbol{c} = 0$
**forall** $\boldsymbol{v} \in D$ **do**
  Initialize $\boldsymbol{v}^{(0)} \leftarrow \boldsymbol{v}$
  **for** $t = 0, ..., k$ **do**
    **for** $i = 1, ..., n$ **do**
      sample $h_i^{(t)} \sim p(h_i|\boldsymbol{v}^{(t)})$
    **end**
    **for** $j = 1, ..., m$ **do**
      sample $v_j^{(t+1)} \sim p(v_j|\boldsymbol{h}^{(t)})$
    **end**
  **end**
  $\mathbf{W} \leftarrow \mathbf{W} + \eta[\sigma(\mathbf{W}\boldsymbol{v}_d + \boldsymbol{c})\boldsymbol{v}_d^T - \sigma(\mathbf{W}\tilde{\boldsymbol{v}} + \boldsymbol{c})\tilde{\boldsymbol{v}}]$
  $\boldsymbol{b} \leftarrow \boldsymbol{b} + \eta[\boldsymbol{v} - \tilde{\boldsymbol{v}}]$
  $\boldsymbol{c} \leftarrow \boldsymbol{c} + \eta[\sigma(\mathbf{W}\boldsymbol{v} + \boldsymbol{c}) - \sigma(\mathbf{W}\tilde{\boldsymbol{v}} + \boldsymbol{c})]$
**end**

FIGURE 2.5: Conrastive divergence algorithm

# Chapter 3

# Experiment Results and Discussions

## 3.1 Experimental Setup

The system was built with tunable hidden layers and number of Gibbs sampling iterations (k). It has been observed that keeping k = 1 is sufficient to gain good performance and that varying it showed no visible improvement in the plots. A binary-binary RBM has been (BB-RBM) used, to perform a pattern storage task.

## 3.2 Database Description

The standard MNIST handwriting dataset was used, quantised to a binary level to emulate a binary random data set of 784 dimensions.

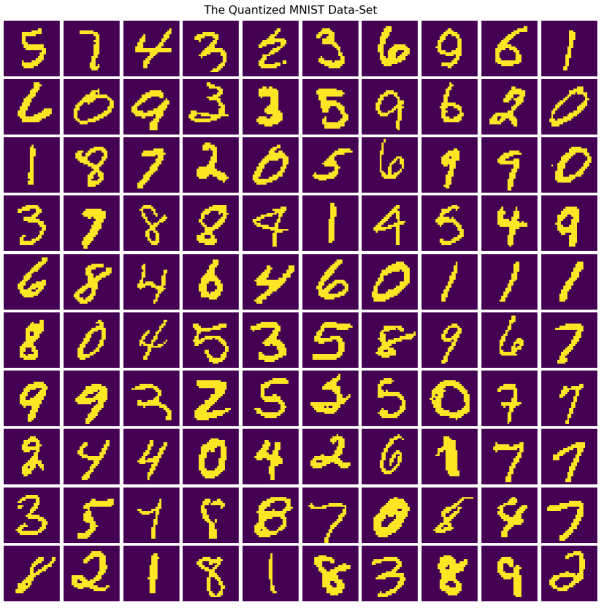


FIGURE 3.1: The Quantized MNIST dataset

## 3.3 Results and Discussions

### 3.3.1 Using a 100 Hidden Layer Neurons

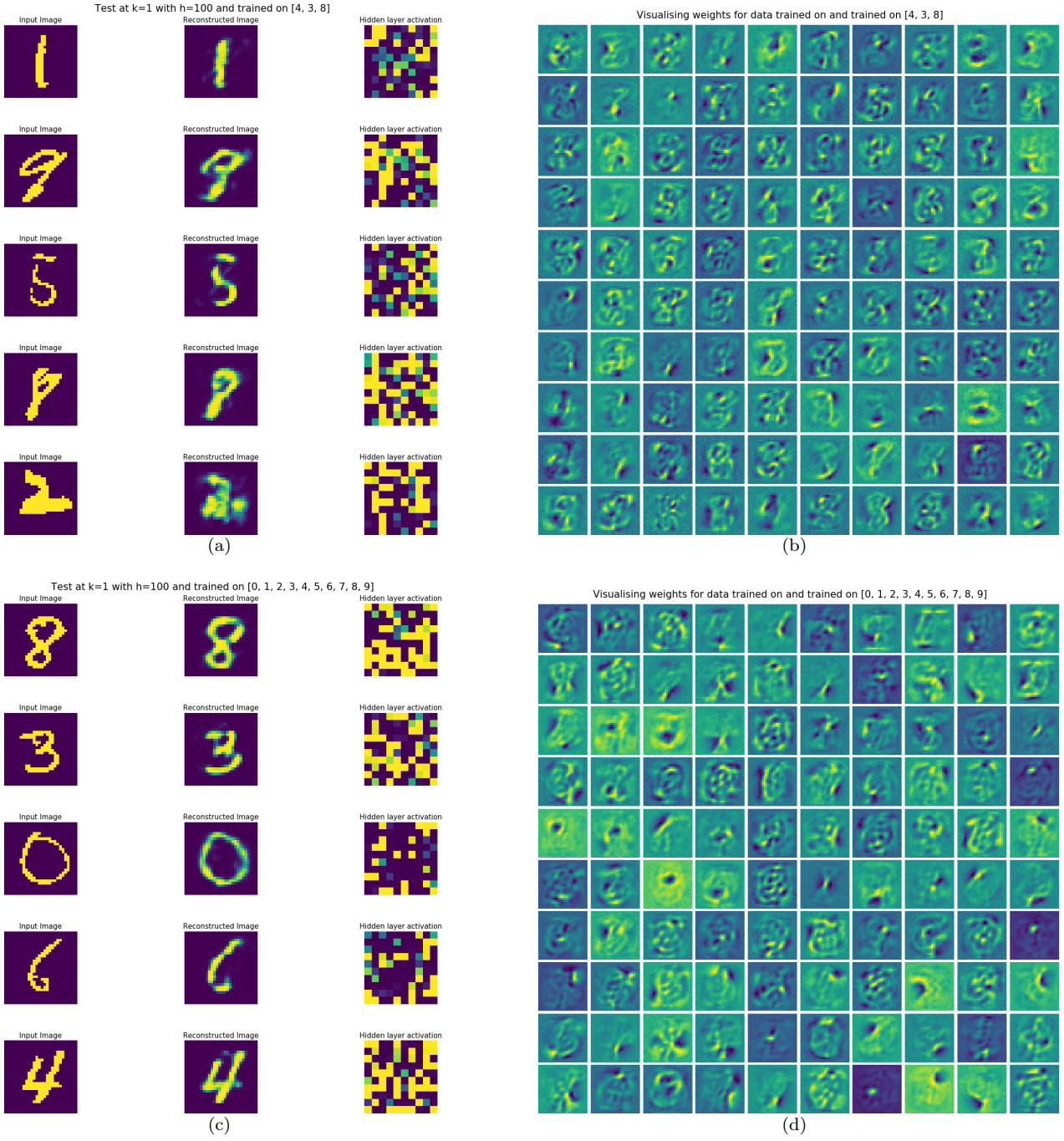


FIGURE 3.2: Figures (a) and (c) show the outputs of an RBM trained on the digits [3,4,8] and [0-9], respectively. Figures (b) and (d) show the weights between inputs and hidden layers, as an image, for these two cases.

### 3.3.2 Using 49 Hidden Layer Neurons

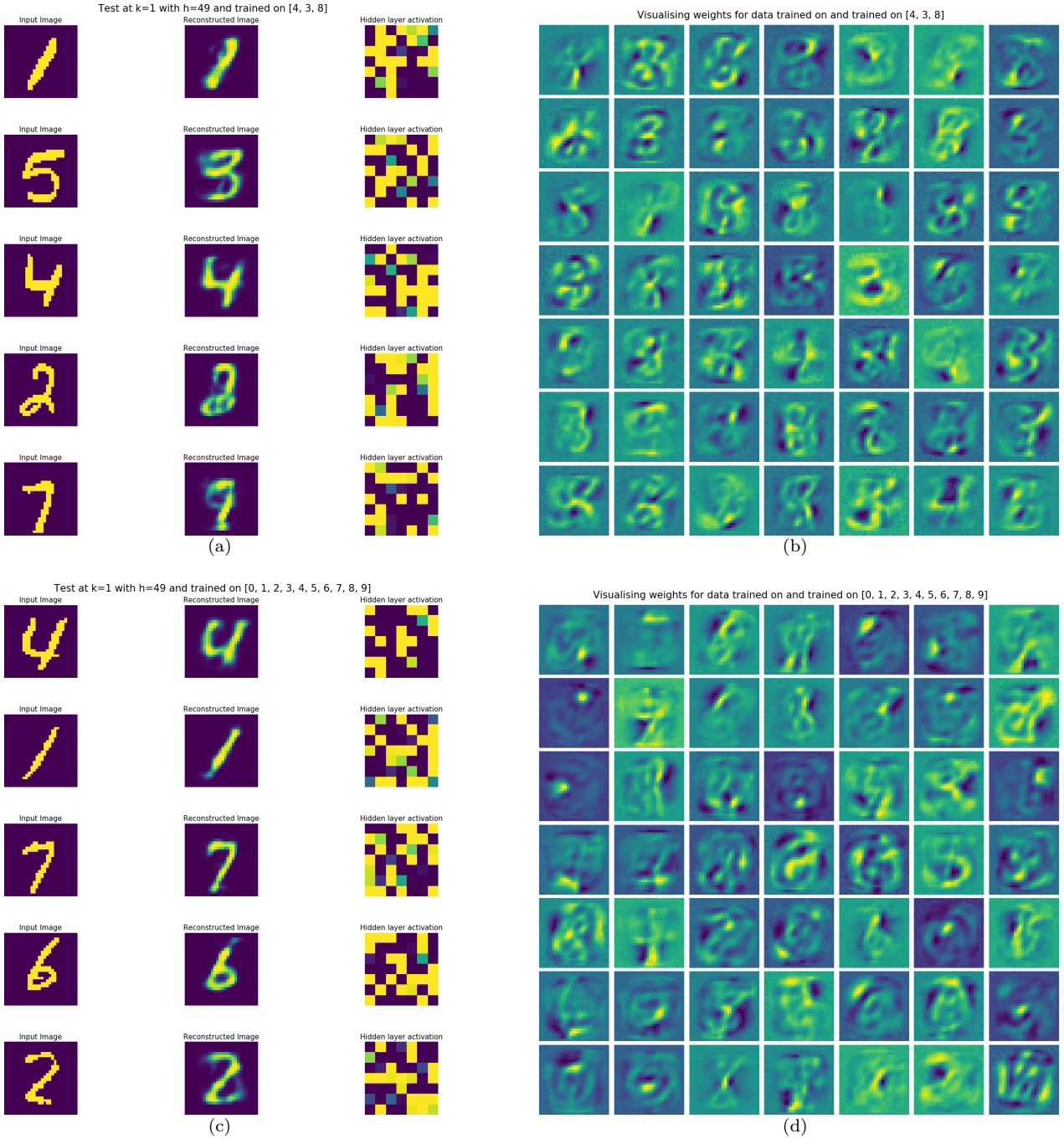


FIGURE 3.3: Figures (a) and (c) show the outputs of an RBM trained on the digits [3,4,8] and [0-9], respectively. Figures (b) and (d) show the weights between inputs and hidden layers, as an image, for these two cases.

### 3.3.3 Using 9 Hidden Layer Neurons

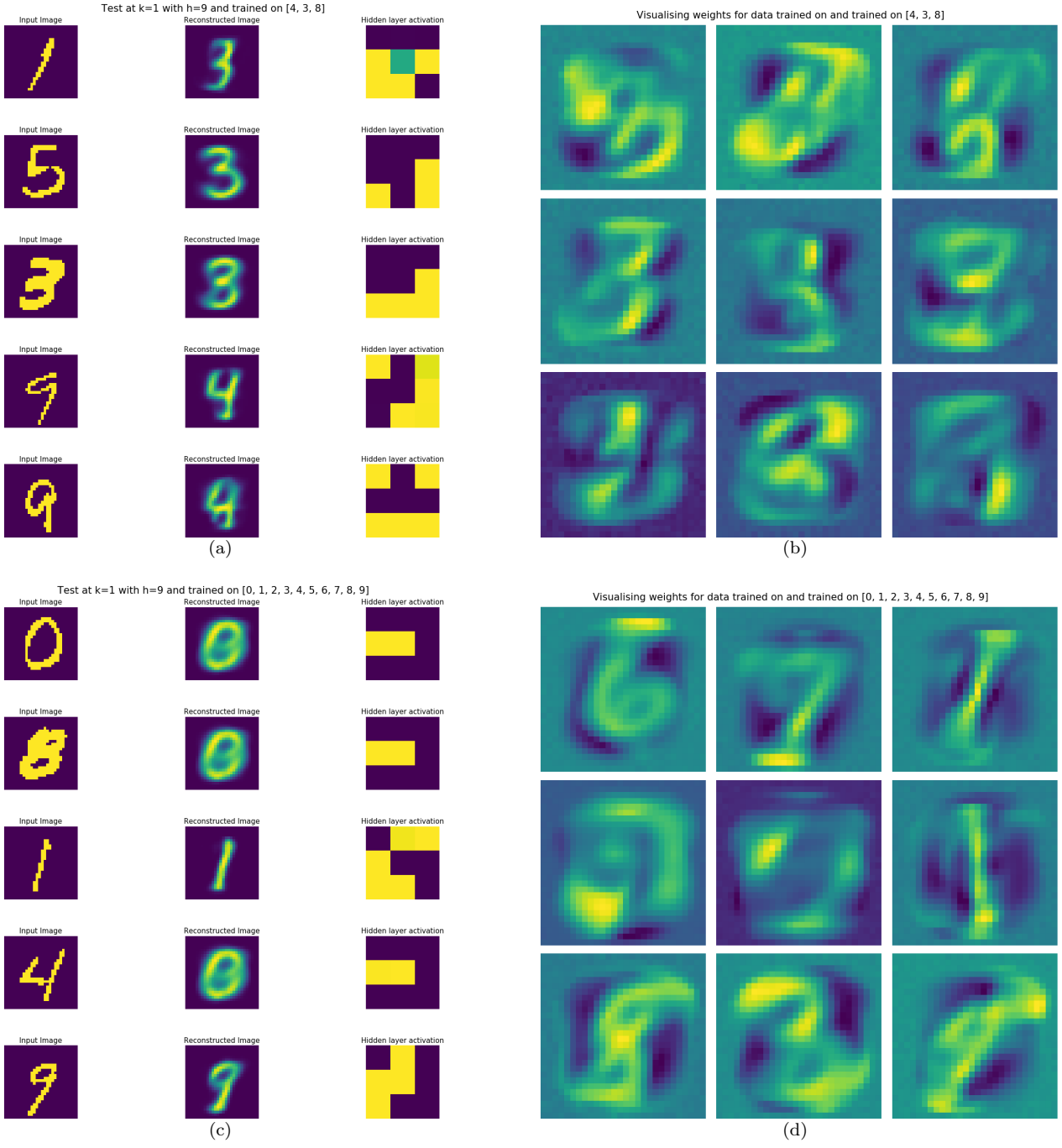


FIGURE 3.4: Figures (a) and (c) show the outputs of an RBM trained on the digits [3,4,8] and [0-9], respectively. Figures (b) and (d) show the weights between inputs and hidden layers, as an image, for these two cases.

# Bibliography

[1] B. Yegnanarayana, *Artificial neural networks.* PHI Learning Pvt. Ltd., 2009.