Universitat Politècnica de Catalunya

Escola Tècnica Superior d'Enginyeria de Telecomunicació de Barcelona

Bachelor's Degree in Engineering Physics

Implementation of Restricted Boltzmann Machines based on archetype selection

Final Bachelor's Thesis

Author:

Marc Fernández Martínez

Supervisor:

Fernando Pablo Mazzanti Castrillejo

Co-Supervisor:

Enrique Romero Merino

May 15, 2023

Contents

Abstract

Restricted Boltzmann Machines are a type of Artificial Neural Network that are used in probability density estimation or classification. In this study, we want to test the ability to generalize a concept of an RBM and understand the limitations that appear when the data provided to train the system is incomplete or noisy (i.e., contaminated with random noise).

In particular, we explore an alternative RBM model based on archetype (ARCH) learning, described in "The emergence of a concept in shallow neural networks" [1], by Elena Agliari, Francesco Alemanno, Adriano Barra and Giordano De Marzo. We want to compare this alternative model with the standard implementation of the network designed for classification problems.

Keywords: Restricted Boltzmann machines; Neural Networks; Gibbs sampling; Constrastive Divergence learning

Acknowledgements

I would like to express my sincere gratitude to my supervisors, Ferran Mazzanti and Enrique Romero, who have supervised this project from the beginning, giving me essential guidelines and support throughout the development of this research.

And finally thank my family, without their constant support it would not have been possible.

Introduction

The use of Deep Learning algorithms, or more specifically, the use of *Neural Networks* for solving complex problems has caught the attention of an increasing number of people in recent years [2].

They excel over classical algorithms in areas where machine learning has driven significant progress, such as computer vision, where *Convolutional Neural Networks* are able to identify facial features and other elements in pictures. This is due to their ability to adapt and learn, to generalise, or to classify data in order to provide solutions to a wide variety of problems [3]. Some *Neural Networks* are even capable of discovering new algorithms that are more efficient in calculating matrix multiplications than current ones [4].

Despite its rise in popularity and the notable advances in recent years, these ideas have been around for more than 60 years and they were first introduced as models of biological neurons that could perform computational tasks [5]. The interest re-emerged with the development of more advanced hardware due to the increased processing power, and new, more ambitious theoretical models were devised. One of them being the Restricted Boltzmann Machine, the model this work focuses on, presented in 1986 with the name *Harmonium* [6]. However, it was not until 2006 that these models were used together and trained with the *Contrastive Divergence* algorithm [7] to build the *Deep Belief Network*; one of the pillars of Deep Learning.

While RBMs are very versatile and can be used for a variety of tasks in machine learning, including dimensionality reduction [8], collaborative filtering [9] and feature learning [10], they also have some limitations. One of the main challenges of RBMs is the computational cost of training them, which can be prohibitive for large datasets

or complex models, due to the fact that the process requires the evaluation of the Partition function at each iteration of the algorithm. Another issue is that RBMs are purely generative models; that is, in order to use them as classifiers, one has to use non-standard implementations such as the Supervised Restricted Boltzmann Machine (SRBM).

In this project we want to explore an alternative setup for the Boltzmann machine which involves training the system based on archetype learning, using a different method to calculate the maximum likelihood parameters by clamping each training example with its class, which makes it able to be used as both for classification and conditional generation. This thesis will focus on trying to reproduce the results stated in [1], and compare the performance of the proposed model to that of the standard RBM.

State of the art and related work

The research in the field of Restricted Boltzmann Machines (RBMs) neural networks has advanced significantly in the recent years, with a focus on developing more powerful and efficient architectures, as well as new training algorithms.

One important step forward was the use of deep belief networks (DBNs), which are made up of multiple layers of RBMs stacked on top of each other [11]. DBNs have been used for a wide range of tasks, such as image recognition, natural language processing, and speech recognition. Additionally, RBMs have been used as building blocks in more complex models such as Generative Adversarial Networks (GANs) and Variational Autoencoders (VAEs) which have been used to generate realistic images, video, and audio.

Another important contribution to the field was the development of the contrastive divergence (CD) and the persistent contrastive divergence (PCD) algorithms for training RBMs. These can significantly reduce the number of iterations required to train RBMs and improve the quality of the generated samples [12]. Additionally,

there has been a lot of research on the use of RBMs for unsupervised learning, which involves training RBMs to extract useful features from raw data that can afterwards be used as inputs to other machine learning algorithms, if required.

Finally and from a purely computational perspective, the popularization of GPUs has been capital to significantly improving the training time of RBMs [13]. Overall, the state of the art for RBMs is constantly evolving, and new developments in the field are likely to continue in the future.

Outline and main contributions

As we have previously mentioned, the main purpose of this thesis is to present an in-depth exploration of Restricted Boltzmann Machines (RBMs), a type of artificial neural network normally used for unsupervised learning, trained using an alternative method proposed by another research group. We start this report by posing the following question.

Question 1 *What are Restricted Boltzmann Machines? How do they work and how are they implemented? How can an RBM be modified in order to use it as a classifier?*

We will focus on this question in Chapter 1, reviewing the RBM neural network and describing what their architecture is. We also introduce the underlying mathematics and statistical principles that govern RBMs.

Next on, we introduce the actual motivation of the research carried out in this project by addressing the ideas behind archetype-based learning. This part of the investigation is the base for the rest of the thesis, as it introduces non-standard concepts in the field of RBMs, and it provides an answer to the following question.

Question 2 *What are the main differences between the standard implementation of an RBM and this archetype-based RBM? What is an archetype in the first place? How can these concepts be used to train a neural network to classify patterns?*

We address this question through Chapter 2, where we develop an RBM model based on [1] that will be used for the later parts of this report.

The next topic that we will focus on is testing our implementation of archetype (ARCH) learning. In particular, we pose the following question.

Question 3 *How can the performance of an ARCH RBM be measured? Can we obtain similar results by reproducing the experiments described in the original work [1]?*

The goal in Chapter 3 is to obtain (at least) similar results, in particular, checking whether the threshold for which the neural network learns the archetypes hold for various measurements, and what are the specific parameters used for each experiment.

Lastly, we can compare the performance of the previous model with a Supervised Restricted Boltzmann Machine (SRBM) when confronted with the same experiments. For that, we formulate the last research question.

Question 4 *Does an ARCH RBM operate in the same way as the SRBM? If so, does the ARCH RBM perform better than the SRBM? How can it be measured?*

To address this question, in Chapter 4 we implement an SRBM and repeat the same experiments as in Chapter 3, so as to have a metric to compare both neural networks. The latter part of this question is answered in Chapter 5, where we discuss whether the ARCH implementation could be used in real problems to learn from large and complex datasets.

Chapter 1

Theoretical Background on Restricted Boltzmann Machines

1.1 Restricted Boltzmann Machines

A Restricted Boltzmann Machine (RBM) is an energy based, undirected graphical model that consists of n visible units $\boldsymbol{V} = (V_1, ..., V_n)$ representing the observable data, and m hidden units $\boldsymbol{H} = (H_1, ..., H_m)$ that capture the correlations between the observed variables [14]. It differs from the more general Boltzmann Machine model in that it adds a restriction such that there cannot be weights (or connections) between units in the same layer.

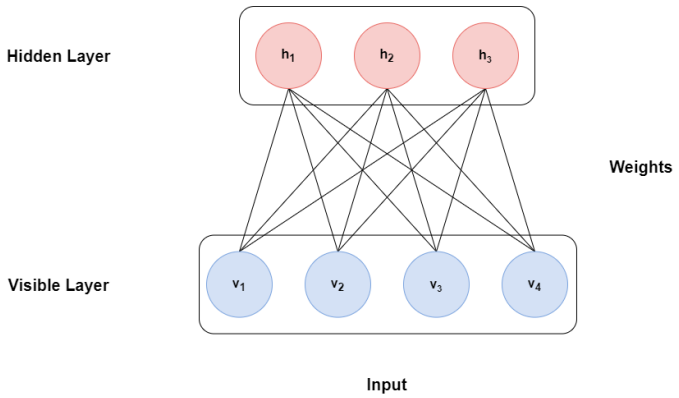


Figure 1: Diagram of an RBM.

In other words, its associated undirected graph is bipartite and its energy function is defined as

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{v}^T \boldsymbol{W} \boldsymbol{h} - \boldsymbol{v}^T \boldsymbol{b} - \boldsymbol{c}^T \boldsymbol{h}. \tag{1.1}$$

We will generally refer to the set $\{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$ of parameters as $\boldsymbol{\theta}$. These can be divided into two groups. The *offsets* or *bias*, composed by the vectors $\boldsymbol{b}$ and $\boldsymbol{c}$, where b_i and c_j are real valued terms associated with the i-th visible and j-th hidden variables, respectively. The second group corresponds to the weight matrix $\boldsymbol{W}$, where each element w_{ij} connects one visible unit v_i to one hidden unit h_j.

We can make a simile with statistical mechanics to derive the expression of the probability of a state of an RBM [15]. Consider a system with a set of states $(\boldsymbol{v}, \boldsymbol{h})$, each with an energy $E(\boldsymbol{v}, \boldsymbol{h})$. We know that, if the system is at a temperature $T > 0$, its state and therefore its average energy will fluctuate over time. After some transient period, the system will reach thermal equilibrium, fluctuating around its average values according to the Boltzmann probability distribution.

In this way, the important result that we extract from physics is that, once at thermal equilibrium, each possible state has probability

$$\mathcal{P}(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} e^{-E(\boldsymbol{v}, \boldsymbol{h})/k_B T}, \tag{1.2}$$

where the normalizing factor

$$Z = \sum_{\boldsymbol{v}, \boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})/k_B T} \tag{1.3}$$

is called the *partition function.*

Equation 1.2 is called the Boltzmann-Gibbs distribution [16] for a system at temperature T. In the case of the RBM, where $\{\boldsymbol{W}, \boldsymbol{b}, \boldsymbol{c}\}$ take values resulting from training a dataset, the temperature is simply a parameter that is actually reabsorbed, so one can safely set $k_B T = 1$.

Since the RBM only has connections between different layers, unit values in different layers become independent variables and therefore the probabilities factorize:

$$p(\boldsymbol{h}|\boldsymbol{v}) = \prod_{j=1}^{m} p(h_j|\boldsymbol{v}) \quad \text{and} \quad p(\boldsymbol{v}|\boldsymbol{h}) = \prod_{i=1}^{n} p(v_i|\boldsymbol{h}). \tag{1.4}$$

Furthermore, and for the same reasons, the unnormalized probability of the model can also be treated and simplified, so that the marginal distribution over the visible variables becomes

$$\begin{aligned} p(\boldsymbol{v}) &= \sum_{\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) = \frac{1}{Z}\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} = \frac{1}{Z}\sum_{h_1}\dots\sum_{h_m} e^{\boldsymbol{v}^T\boldsymbol{b}}\prod_{j=1}^{m} e^{h_j(c_j+\boldsymbol{v}\boldsymbol{W}_i)} \\ &= \frac{1}{Z}e^{\boldsymbol{v}^T\boldsymbol{b}}\prod_{j=1}^{m}\sum_{h_j} e^{h_j(c_j+\boldsymbol{v}^T\boldsymbol{W}_i)} = \frac{1}{Z}e^{\boldsymbol{v}^T\boldsymbol{b}}\prod_{j=1}^{m}(e^{-c_j-\boldsymbol{v}^T\boldsymbol{W}_i}+e^{c_j+\boldsymbol{v}^T\boldsymbol{W}_i}). \end{aligned} \tag{1.5}$$

In the last step, we made use of the fact that we use binary units $v_i, h_j \in \{-1,1\}, \forall i \in [1,n], \forall j \in [1,m]$.

We can develop the expression in eq.1.4. Let $\boldsymbol{v}_{-\ell}$ denote the state of all visible units except the ℓ-th one and let us define

$$\begin{aligned} \alpha_\ell(\boldsymbol{h}) &= -\sum_{j=1}^{m} w_{\ell j}h_j - b_\ell \\ \beta(\boldsymbol{v}_{-\ell},\boldsymbol{h}) &= -\sum_{i=1,i\neq\ell}^{n}\sum_{j=1}^{m} w_{ij}v_ih_j - \sum_{i=1,i\neq\ell}^{n} b_iv_i - \sum_{j=1}^{m} c_jh_j. \end{aligned}$$

Using eq.1.1, we find that $E(\boldsymbol{v},\boldsymbol{h}) = \beta(\boldsymbol{v}_{-\ell},\boldsymbol{h}) + v_\ell\alpha_\ell(\boldsymbol{h})$, and write

$$\begin{aligned} p(v_\ell|\boldsymbol{h}) &= p(v_\ell = 1|\boldsymbol{v}_{-\ell},\boldsymbol{h}) = \frac{p(v_\ell=1,\boldsymbol{v}_{-\ell},\boldsymbol{h})}{p(\boldsymbol{v}_{-\ell},\boldsymbol{h})} \\ &= \frac{e^{-E(v_\ell=1,\boldsymbol{v}_{-\ell},\boldsymbol{h})}}{e^{-E(v_\ell=-1,\boldsymbol{v}_{-\ell},\boldsymbol{h})}+e^{-E(v_\ell=1,\boldsymbol{v}_{-\ell},\boldsymbol{h})}} = \frac{e^{-\beta(\boldsymbol{v}_{-\ell},\boldsymbol{h})-1\cdot\alpha_\ell(\boldsymbol{h})}}{e^{-\beta(\boldsymbol{v}_{-\ell},\boldsymbol{h})+1\cdot\alpha_\ell(\boldsymbol{h})}+e^{-\beta(\boldsymbol{v}_{-\ell},\boldsymbol{h})-1\cdot\alpha_\ell(\boldsymbol{h})}} \\ &= \frac{e^{-\beta(\boldsymbol{v}_{-\ell},\boldsymbol{h})}e^{-\alpha_\ell(\boldsymbol{h})}}{e^{-\beta(\boldsymbol{v}_{-\ell},\boldsymbol{h})}e^{\alpha_\ell(\boldsymbol{h})}+e^{-\beta(\boldsymbol{v}_{-\ell},\boldsymbol{h})}e^{-\alpha_\ell(\boldsymbol{h})}} = \frac{e^{-\beta(\boldsymbol{v}_{-\ell},\boldsymbol{h})}e^{-\alpha_\ell(\boldsymbol{h})}}{e^{-\beta(\boldsymbol{v}_{-\ell},\boldsymbol{h})}\left(e^{\alpha_\ell(\boldsymbol{h})}+e^{-\alpha_\ell(\boldsymbol{h})}\right)} \\ &= \frac{e^{-\alpha_\ell(\boldsymbol{h})}}{e^{-\alpha_\ell(\boldsymbol{h})}+e^{\alpha_\ell(\boldsymbol{h})}} = \frac{1}{2}\left(1+\frac{e^{-\alpha_\ell(\boldsymbol{h})}-e^{\alpha_\ell(\boldsymbol{h})}}{e^{-\alpha_\ell(\boldsymbol{h})}+e^{\alpha_\ell(\boldsymbol{h})}}\right) \\ &= \frac{1}{2}\left(1+\tanh\left(-\alpha_\ell(\boldsymbol{h})\right)\right) = \frac{1}{2}(1+\tanh\left(\sum_{j=1}^{m} w_{\ell j}h_j + b_\ell\right)). \end{aligned} \tag{1.6}$$

We can find the expression for $p(h_j = 1|\boldsymbol{v})$ in a similar manner. Let $\boldsymbol{h}_{-\ell}$ denote the

state of all hidden units except the ℓ-th one and define

$$\gamma_\ell(\boldsymbol{v}) = -\sum_{i=1}^{n} w_{i\ell} v_i - c_\ell$$
$$\delta(\boldsymbol{h}_{-\ell}, \boldsymbol{v}) = -\sum_{i=1}^{n}\sum_{j=1, j\neq\ell}^{m} w_{ij} v_i h_j - \sum_{i=1}^{n} b_i v_i - \sum_{j=1, j\neq\ell}^{m} c_j h_j.$$

Then, proceeding as before, we write

$$\begin{aligned} p(h_\ell|\boldsymbol{v}) = p(h_\ell = 1|\boldsymbol{h}_{-\ell}, \boldsymbol{v}) &= \frac{p(h_\ell = 1, \boldsymbol{h}_{-\ell}, \boldsymbol{v})}{p(h_{-\ell}, \boldsymbol{v})} \\ &= \frac{e^{-E(h_\ell=1, \boldsymbol{h}_{-\ell}, \boldsymbol{v})}}{e^{-E(h_\ell=-1, \boldsymbol{h}_{-\ell}, \boldsymbol{v})} + e^{-E(h_\ell=1, \boldsymbol{h}_{-\ell}, \boldsymbol{v})}} = \frac{e^{-\delta(\boldsymbol{h}_{-\ell}, \boldsymbol{v}) - 1\cdot\gamma_\ell(\boldsymbol{h})}}{e^{-\delta(\boldsymbol{h}_{-\ell}, \boldsymbol{b}) + 1\cdot\gamma_\ell(\boldsymbol{v})} + e^{-\delta(\boldsymbol{h}_{-\ell}, \boldsymbol{v}) - 1\cdot\gamma_\ell(\boldsymbol{v})}} \\ &= \frac{e^{-\delta(\boldsymbol{h}_{-\ell}, \boldsymbol{v})} e^{-\gamma_\ell(\boldsymbol{v})}}{e^{-\delta(\boldsymbol{h}_{-\ell}, \boldsymbol{v})} e^{\gamma_\ell(\boldsymbol{v})} + e^{-\delta(\boldsymbol{h}_{-\ell}, \boldsymbol{v})} e^{-\gamma_\ell(\boldsymbol{v})}} = \frac{e^{-\delta(\boldsymbol{h}_{-\ell}, \boldsymbol{v})} e^{-\gamma_\ell(\boldsymbol{v})}}{e^{-\delta(\boldsymbol{h}_{-\ell}, \boldsymbol{v})}\left(e^{\gamma_\ell(\boldsymbol{v})} + e^{-\gamma_\ell(\boldsymbol{v})}\right)} \\ &= \frac{e^{-\gamma_\ell(\boldsymbol{v})}}{e^{-\gamma_\ell(\boldsymbol{h})} + e^{\gamma_\ell(\boldsymbol{v})}} = \frac{1}{2}(1 + \tanh\left(\sum_{i=1}^{n} w_{i\ell} v_i + c_\ell\right)) \end{aligned} \tag{1.7}$$

1.2 Gradient ascent formulation

The main idea behind unsupervised learning in neural networks is to extract and learn the hidden features characterizing a probability distribution $q(\boldsymbol{v})$ based on data sampled from it. Knowing the structure of the network and assuming its energy function is parameterized as in eq.1.1, learning from the training data means to adjust the value of $\boldsymbol{\theta}$ so as to make the model probabilities match as accurately as possible to $q(\boldsymbol{v})$. In general, it is not possible to find the optimal parameters analytically, and numerical methods need to be used in order to obtain good approximations. However, there are measures to determine how close these parameters are to the optimal values.

1.2.1 Gradient of the log-likelihood

Consider a training dataset (TSet for short) $\mathcal{S} = \boldsymbol{v}_1, ..., \boldsymbol{v}_\ell$, where all samples are assumed to be independent and drawn from the same distribution $q(\boldsymbol{v})$. Then, a

standard way of estimating $\boldsymbol{\theta}$ is to optimize the *maximum-likelihood estimator*. This is done finding the parameters that maximize the probability of $\mathcal{S}$ over the model distribution.

$$\mathcal{L}(\boldsymbol{\theta}|\mathcal{S}) = \prod_{i=1}^{\ell} p(\boldsymbol{v}_i|\boldsymbol{\theta}), \tag{1.8}$$

which is equivalent as maximizing its logarithm

$$\ln \mathcal{L}(\boldsymbol{\theta}|\mathcal{S}) = \ln \prod_{i=1}^{\ell} p(\boldsymbol{v}_i|\boldsymbol{\theta}) = \sum_{i=1}^{\ell} \ln p(\boldsymbol{v}_i|\boldsymbol{\theta}). \tag{1.9}$$

Equivalently, if we assume that the empirical distribution of the training set is uniform (i.e., $q(\boldsymbol{v}) = \frac{1}{\ell}$, where ℓ is the size of the dataset), we can minimize its distance with our model distribution $p(\boldsymbol{v})$ in terms of the *Kullback-Leibler divergence* (KL Divergence) [17], which is defined as

$$\begin{aligned} KL(q||p) &= \sum_{\boldsymbol{v}\in\mathcal{S}} q(\boldsymbol{v}) \ln\left(\frac{q(\boldsymbol{v})}{p(\boldsymbol{v})}\right) = \sum_{\boldsymbol{v}\in\mathcal{S}} q(\boldsymbol{v}) \ln(q(\boldsymbol{v})) - \sum_{\boldsymbol{v}\in\mathcal{S}} q(\boldsymbol{v}) \ln(p(\boldsymbol{v})) \\ &= -\ln(\ell) - \frac{1}{\ell} \ln \mathcal{L}(\boldsymbol{\theta}|\mathcal{S}). \end{aligned} \tag{1.10}$$

Note that eq.1.10 can be seen as the difference between the entropy of $q(\boldsymbol{v})$ and a second term. The latter becomes the log-likelihood when approximating this term by the training samples in $\mathcal{S}$ and assuming $q(\boldsymbol{v})$ is uniform. Thus, maximizing the log-likelihood corresponds to minimizing the KL.

As mentioned above, it is not possible in general to find the optimal parameters analytically. The standard way to tackle this problem is to apply gradient ascent on the log-likelihood. Based on general grounds, one can consider the following update rule

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \epsilon f'(\boldsymbol{\theta}^{(t)}), \tag{1.11}$$

where the constant $\epsilon \in \mathbb{R}_0^+$ is called the *learning rate*, and it controls the change in the parameters during each step of the learning process. The other term $f'(\boldsymbol{\theta}^{(t)})$ is

the derivative of the *target function* to maximize. In a plain vanilla gradient ascent scheme, this target function corresponds to the log-likelihood:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \epsilon \frac{\partial}{\partial \boldsymbol{\theta}^{(t)}} (\ln \mathcal{L}(\boldsymbol{\theta}^{(t)} | \mathcal{S})). \tag{1.12}$$

However, one can add another term to the target function

$$f(\boldsymbol{\theta}^{(t)}) = \epsilon \ln \mathcal{L}(\boldsymbol{\theta}^{(t)} | \mathcal{S})) - \frac{\lambda}{2} \left\| \boldsymbol{\theta}^{(t)} \right\|^2. \tag{1.13}$$

Now, maximizing the expression in eq.1.13 would also minimize the norm of the parameters, scaled by the constant $\lambda \in \mathbb{R}_0^+$ known as the *weight decay*. This means that during the learning process, the RBM searches for parameters that maximize the log-likelihood while also having small absolute values.

We can introduce one more extension to the gradient ascent optimization algorithm, often referred to as *momentum* that is used to avoid undesirable oscillations in the iterative procedure, thus speeding up the learning process [18]. It is defined as the change used at the previous step ($\boldsymbol{\omega}^{(t-1)} = f'(\boldsymbol{\theta}^{(t-1)})$), weighted by the momentum constant $\mu \in \mathbb{R}_0^+$.

Adding these new terms to eq.1.11, leads to the gradient ascent update rule

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \epsilon \frac{\partial}{\partial \boldsymbol{\theta}^{(t)}} (\ln \mathcal{L}(\boldsymbol{\theta}^{(t)} | \mathcal{S})) - \lambda \boldsymbol{\theta}^{(t)} + \mu \boldsymbol{\omega}^{(t-1)}. \tag{1.14}$$

Finally, we derive the expression of the gradient of the log-likelihood (i.e., the term weighted by the learning rate in eq.1.14) for each of the parameters of the RBM. The general expression for the gradient with respect to $\boldsymbol{\theta}$ given a single training example $\boldsymbol{v}$ is

$$\begin{aligned} \frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}, \boldsymbol{v})}{\partial \boldsymbol{\theta}} &= \frac{\partial}{\partial \boldsymbol{\theta}} \left(\ln \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \right) - \frac{\partial}{\partial \boldsymbol{\theta}} \left(\ln \sum_{\tilde{\boldsymbol{v}}, \boldsymbol{h}} e^{-E(\tilde{\boldsymbol{v}}, \boldsymbol{h})} \right) \\ &= - \frac{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial \boldsymbol{\theta}}}{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})}} + \frac{\sum_{\tilde{\boldsymbol{v}}, \boldsymbol{h}} e^{-E(\tilde{\boldsymbol{v}}, \boldsymbol{h})} \frac{\partial E(\tilde{\boldsymbol{v}}, \boldsymbol{h})}{\partial \boldsymbol{\theta}}}{\sum_{\tilde{\boldsymbol{v}}, \boldsymbol{h}} e^{-E(\tilde{\boldsymbol{v}}, \boldsymbol{h})}} \end{aligned}$$

$$= -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v})\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} + \sum_{\tilde{\boldsymbol{v}},\boldsymbol{h}} p(\tilde{\boldsymbol{v}},\boldsymbol{h})\frac{\partial E(\tilde{\boldsymbol{v}},\boldsymbol{h})}{\partial \boldsymbol{\theta}}. \tag{1.15}$$

Where we used that the conditional probability can be written

$$p(\boldsymbol{h}|\boldsymbol{v}) = \frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{v})} = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}.$$

Also, note how the first term of eq.1.15 is calculated under the distribution $p(\boldsymbol{h}|\boldsymbol{v})$, where $\boldsymbol{v}$ corresponds to an example from the training set. This term is known as the *positive phase*, and in the sense of the computational cost, it is cheap to evaluate. On the other hand, the second term evaluates $p(\tilde{\boldsymbol{v}},\boldsymbol{h})$ for all the possible visible configurations $\tilde{\boldsymbol{v}}$ and $\boldsymbol{h}$. This is known as the *negative phase*, and evaluating this sum has a cost that scales exponentially with the size of the smaller layer (i.e., $\mathcal{O}(2^n)$ or $\mathcal{O}(2^m)$, where n and m are the number of visible and hidden units, respectively), as one can make use of a clever factorization trick on either the hidden or the visible variables.

To show an example of this factorization, the derivative of the log-likelihood of a single training pattern $\boldsymbol{v}$ w.r.t. the weight w_{ij} is

$$\begin{aligned}
\frac{\partial \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial w_{ij}} &= -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v})\frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial w_{ij}} + \sum_{\tilde{\boldsymbol{v}},\boldsymbol{h}} p(\tilde{\boldsymbol{v}},\boldsymbol{h})\frac{\partial E(\tilde{\boldsymbol{v}},\boldsymbol{h})}{\partial w_{ij}} \\
&= \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) v_i h_j - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}}) \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\tilde{\boldsymbol{v}}) \tilde{v}_i h_j \\
&= \sum_{\boldsymbol{h}} \prod_{k=1}^{m} p(h_k|\boldsymbol{v}) v_i h_j - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}}) \sum_{\boldsymbol{h}} \prod_{k=1}^{m} p(h_k|\tilde{\boldsymbol{v}}) \tilde{v}_i h_j \\
&= \sum_{h_j} \sum_{\boldsymbol{h}_{-j}} p(h_j|\boldsymbol{v}) p(\boldsymbol{h}_{-j}|\boldsymbol{v}) v_i h_j - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}}) \sum_{h_j} \sum_{\boldsymbol{h}_{-j}} p(h_j|\tilde{\boldsymbol{v}}) p(\boldsymbol{h}_{-j}|\tilde{\boldsymbol{v}}) \tilde{v}_i h_j \\
&= \sum_{h_j} p(h_j|\boldsymbol{v}) v_i h_j \underbrace{\sum_{\boldsymbol{h}_{-j}} p(\boldsymbol{h}_{-j}|\boldsymbol{v})}_{=1} - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}}) \sum_{h_j} p(h_j|\tilde{\boldsymbol{v}}) \tilde{v}_i h_j \underbrace{\sum_{\boldsymbol{h}_{-j}} p(\boldsymbol{h}_{-j}|\tilde{\boldsymbol{v}})}_{=1} \\
&= \sum_{h_j} p(h_j|\boldsymbol{v}) v_i h_j - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}}) \sum_{h_j} p(h_j|\tilde{\boldsymbol{v}}) \tilde{v}_i h_j \\
&= (-p(h_j = -1|\boldsymbol{v}) + p(h_j = 1|\boldsymbol{v})) v_i - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}}) (-p(h_j = -1|\tilde{\boldsymbol{v}}) + p(h_j = 1|\tilde{\boldsymbol{v}})) \tilde{v}_i
\end{aligned}$$

$$= (2p(h_j = 1|\boldsymbol{v}) - 1)v_i - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}})(2p(h_j = 1|\tilde{\boldsymbol{v}}) - 1)\tilde{v}_i$$

$$= \tanh\left(\sum_{i=1}^{n} w_{ij}v_i - c_j\right) v_i - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}}) \tanh\left(\sum_{i=1}^{n} w_{ij}\tilde{v}_i - c_j\right) \tilde{v}_i. \tag{1.16}$$

We denoted all hidden variables except the j-th one as $\boldsymbol{h}_{-j}$, and factorized the sum for the visible layer.

Analogously to eq.1.16 we can find the derivatives w.r.t. the bias b_i of the i-th visible variable

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial b_i} = v_i - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}})\tilde{v}_i \tag{1.17}$$

and w.r.t. the bias parameter c_j of the j-th hidden variable

$$\frac{\partial \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial c_j} = \tanh\left(\sum_{i=1}^{n} w_{ij}v_i - c_j\right) - \sum_{\tilde{\boldsymbol{v}}} p(\tilde{\boldsymbol{v}}) \tanh\left(\sum_{i=1}^{n} w_{ij}\tilde{v}_i - c_j\right). \tag{1.18}$$

The full proof for these expressions is found in the appendix.

1.2.2 Mini-Batch Gradient Descent

One of the best ways to obtain a high performance machine learning system, is to take an efficient learning algorithm and train it on a lot of data. The issue with training on a large dataset, is that computing the gradient for a single step of gradient ascent becomes very expensive computationally due to the need to sum over all of its terms. Only after having cycled through the entire dataset, which is known as an *epoch*, does one perform an update of the model parameters. The solution is quite simple: split the whole training set in groups or *batches*, and update the parameters after each batch is processed.

In this way, mini-batch gradient ascent uses a parameter b called the *mini batch size* to select the number of examples that are going to be used in each epoch of the algorithm. Specifically, for each iteration we perform the following steps:

1. Generate a random permutation of the training set. In practice, this helps

speed up the convergence by breaking the symmetries of always visiting the examples in the same order.

2. Group the examples in batches of size b (including one of size = number examples mod b in case it is not divisible by b).

3. For each mini-batch, update the weights and bias according to the learning rules of section 1.2.

The value of b can change depending on the problem, but the extreme cases are $b = 1$ and $b = \ell$, ℓ being the size of the training set.

In the first case, known as *Online Gradient Ascent* (OGD), the parameters of the RBM are updated after each training state is visited, making the computation robust to large datasets, but requiring more epochs to arrive at the maximum [19]. Alternatively, one can use *Batch Gradient Ascent* and update the parameters only after running through the whole dataset. In this case, the algorithm reaches the maximum in less steps, but the large amount of calculations that have to be performed at each epoch can slow the overall process significantly when the training set is very large, as mentioned before.

The middle point between those two cases is *Mini-Batch Gradient Ascent*, which can be even faster than OGD by using vectorization in order to compute the gradients, in other words, applying the operations to the entire mini-batch at once [20]. An example of mini-batch gradient ascent using vectorization on a modern programming language (Julia) can be found in the appendix.

In the next section we will introduce a method for approximating the *negative phase*.

1.3 Learning algorithm

1.3.1 Gibbs sampling

Gibbs sampling is a Monte Carlo Markov Chain algorithm used to approximate a sequence of observations from a multivariate joint probability distribution that is applicable when the joint distribution is not known explicitly or is difficult to sample from directly, but the conditional distribution of each variable is known and easier

to sample from. The idea is to build a Markov chain [21] by updating each variable based on the conditional distribution given the state of the others.

Given a joint distribution of P random variables $\boldsymbol{V} = \{v_1, v_2, ..., v_P\}$, and assuming that the Markov chain changes its state over time, we can consider $\boldsymbol{V}^{(k)} = (v_1^{(k)}, v_2^{(k)}, ..., v_P^{(k)})$ as the state at time $k \geq 0$. Between two successive points in time, in a Gibbs sampling scheme, the new state of the chain is obtained from a sequence of N sub-sampling steps. First, a variable $v_i, i \in [1, P]$ is chosen randomly. Then, the new state for v_i is sampled based on its conditional probability distribution given all the other variables, namely

$$v_i \sim \mathcal{P}(v_i|v_{-i}), \tag{1.19}$$

where we denoted all variables but the i-th as v_{-i}.

The process is repeated P times, one for each variable, thus completing a single step of the Markov chain and obtaining a sample from the joint probability distribution. It can be shown [22] that the Markov chain built in this way converges to the joint distribution of $\boldsymbol{V}$ if the chain is aperiodic and irreducible. In the same book, there is also a proof for the *periodic Gibbs sampler*, in which the variables to be updated are selected in a fixed predefined order (see [23]).

In Restricted Boltzmann Machines, this sampling method is efficiently computed because the conditional probabilities $p(\boldsymbol{v}|\boldsymbol{h})$ and $p(\boldsymbol{h}|\boldsymbol{v})$ can be factorized, as described in the previous sections. Defining the set of variables $\boldsymbol{X} = \{\boldsymbol{v}, \boldsymbol{h}\}$, we start the chain with $\boldsymbol{v}^{(1)}$, a member of the training set. Then, Gibbs sampling is done in 2 steps:

- Sample $\boldsymbol{h}^{(k)}$ from $p(\boldsymbol{h}|\boldsymbol{v}^{(k)})$.
- Then get $\boldsymbol{v}^{(k+1)}$ from $p(\boldsymbol{v}|\boldsymbol{h}^{(k)})$.

1.3.2 Contrastive Divergence

Recalling the expression of the gradient in eq.1.15, the first term or *positive phase*, is efficiently calculated thanks to the factorization of the conditional probabilities and the fact that the averages are only taken over the training set. Sadly, the *negative*

phase, which can be seen as the expected value of the gradient of the energy w.r.t. the parameters (i.e., $\mathbb{E}_{p(\tilde{\boldsymbol{v}},\boldsymbol{h})}[\frac{\partial E(\tilde{\boldsymbol{v}},\boldsymbol{h})}{\partial \boldsymbol{\theta}}] = \mathbb{E}_{p(\tilde{\boldsymbol{v}})}[\mathbb{E}_{p(\boldsymbol{h}|\tilde{\boldsymbol{v}})}[\frac{\partial E(\tilde{\boldsymbol{v}},\boldsymbol{h})}{\partial \boldsymbol{\theta}}]]$), is intractable when the size of the RBM increases as one would have to sum over the complete set of states of the space.

We can reduce the cost of computing by introducing two approximations. Firstly, we limit the length of the chain obtained via Gibbs sampling to $\boldsymbol{k}$ steps. It was shown in [24] that calculating $p(\boldsymbol{h}|\boldsymbol{v})$ with this chain could produce a good enough approximation that improves the larger $\boldsymbol{k}$ is. Then, one replaces the expectation over all the possible states in the space (which would be exponential in the size of the smaller layer), with the value of that expectation obtained from a single sample $\boldsymbol{v}$ taken from the training set. The statistical representation of all states is then expected to be well represented by the corresponding Gibbs sampling exploration obtained from the repeated procedure applied to all members of the training set.

These two approximations define the new update rule for our algorithm, the k-step Contrastive Divergence, or CD-k, used to approximate the negative phase as seen in Alg.1.

Algorithm 1 k-step contrastive divergence

Input: RBM $(\boldsymbol{v}, \boldsymbol{h})$, training batch $\mathcal{S}$.
Output: gradient approximation $\Delta \boldsymbol{W}$, $\Delta \boldsymbol{b}$ and $\Delta \boldsymbol{c}$.

1: init $\Delta \boldsymbol{W} = \boldsymbol{0}_{N_v,N_h}, \Delta \boldsymbol{b} = \boldsymbol{0}_{N_v}, \Delta \boldsymbol{c} = \boldsymbol{0}_{N_h}$.
2: **for all the** $\boldsymbol{v} \in \mathcal{S}$ **do**
3: $\quad \boldsymbol{v}^{(0)} \leftarrow \boldsymbol{v}$
4: $\quad$ **for** $t = 0, ..., k-1$ **do**
5: $\quad\quad$ sample $\boldsymbol{h}^{(t)} \sim p(\boldsymbol{h}|\boldsymbol{v}^{(t)})$
6: $\quad\quad$ sample $\boldsymbol{v}^{(t+1)} \sim p(\boldsymbol{v}|\boldsymbol{h}^{(t)})$
7: $\quad \Delta \boldsymbol{W} \leftarrow \Delta \boldsymbol{W} + \boldsymbol{v}^{(0)} \cdot \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v}^{(0)})}[\boldsymbol{h}]^T - \boldsymbol{v}^{(k)} \cdot \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v}^{(k)})}[\boldsymbol{h}]^T$
8: $\quad \Delta \boldsymbol{b} \leftarrow \Delta \boldsymbol{b} + \boldsymbol{v}^{(0)} - \boldsymbol{v}^{(k)}$
9: $\quad \Delta \boldsymbol{c} \leftarrow \Delta \boldsymbol{c} + \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v}^{(0)})}[\boldsymbol{h}] - \mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v}^{(k)})}[\boldsymbol{h}]$

This algorithm calculates the approximated gradients, which are then added to the momentum and weight decay terms in eq.1.14 to update the values of the parameters $\boldsymbol{\theta}$. In this way, we obtain better results and a higher performance overall compared to the vanilla gradient ascent.

1.3.3 Hyperparameter tuning

The *hyperparameter tuning function* is a function that automatically searches for the optimal values of the hyperparameters for our model, such as the learning rate ϵ, the weight decay λ and the momentum μ seen in eq.1.11, and even the number of hidden neurons N_h.

This method works by iteratively selecting a set of hyperparameters from a defined search space (e.g., a grid of values), training the RBM model with those hyperparameters and evaluating the performance of the model on a smaller dataset called the *validation set*, keeping the set of parameters that yield the best results. For example, we can use this function in order to find the optimal parameters based on the obtained classification accuracy on the validation set. A detailed implementation of this strategy is found in the appendix.

1.4 Supervised Restricted Boltzmann Machines

Supervised Restricted Boltzmann Machines (SRBMs) are a variant of the standard RBM specifically designed for supervised learning tasks where the training data is labeled [25]. In the SRBM, these units corresponding to the class labels are appended to the visible inputs of the network, as seen in Figure 2.

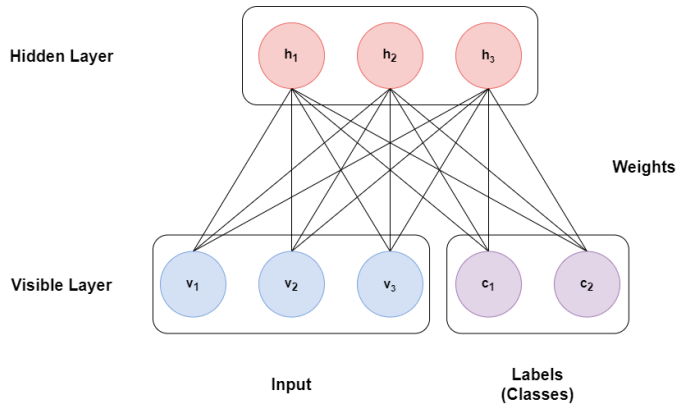


Figure 2: Diagram of an SRBM with 2 encoded labels in the visible layer.

The overall implementation of the SRBM is the same as in the standard RBM that we introduced in Section 1.1, including the use of bias in its energy function.

During training, the model learns to reconstruct the input data by minimizing the difference between the original and the sampled dataset input using Contrastive

Divergence (CD, see Alg.1) as well as predicting the corresponding label. This approach allows the SRBM to capture better the underlying structure of the data (i.e., the features that we want it to learn) and improve its classification performance.

The following equation describes the classification probability when the labels are encoded in the visible layer of an SRBM. Given an input vector $\boldsymbol{v}$, the probability of reaching a correct label $\boldsymbol{y} \in \boldsymbol{Y}$ ($\boldsymbol{Y}$ being the set with all the defined labels of a problem) is calculated following the expression

$$\mathcal{P}(\boldsymbol{y}|\boldsymbol{v}) = \frac{\mathcal{P}(\boldsymbol{v}, \boldsymbol{y})}{\mathcal{P}(\boldsymbol{v})} = \frac{\mathcal{P}(\boldsymbol{v}, \boldsymbol{y})}{\sum_{\boldsymbol{y} \in \boldsymbol{Y}} \mathcal{P}(\boldsymbol{v}, \boldsymbol{y})}. \tag{1.20}$$

$\mathcal{P}(\boldsymbol{v}, \boldsymbol{y})$ is calculated as the marginal probability (see eq.1.5) of the visible input $\boldsymbol{V} = (v_1, \ldots, v_N, y_1, \ldots, y_K)$, where N and K are the lengths of $\boldsymbol{v}$ and $\boldsymbol{y}$ respectively, and the total number of visible units is given by $N_v = N + K$. Calculating this conditional probability is relatively easy in the computational sense, as the sum in the denominator of eq.1.20 runs for the number of labels.

On the other hand, given a label $\boldsymbol{y} \in \boldsymbol{Y}$ the probability of reaching a state $\boldsymbol{v}$ is calculated as

$$\mathcal{P}(\boldsymbol{v}|\boldsymbol{y}) = \frac{\mathcal{P}(\boldsymbol{v}, \boldsymbol{y})}{\sum_{\tilde{\boldsymbol{v}}} \mathcal{P}(\tilde{\boldsymbol{v}}, \boldsymbol{y})}. \tag{1.21}$$

Unfortunately, we cannot simplify the expression, so the sum runs for all possible states $\tilde{\boldsymbol{v}}$ and its computational cost is $\mathcal{O}(2^N)$. This is too expensive to calculate when the length of the input vectors N is relatively large (i.e., $N > 25$).

Chapter 2

Restricted Boltzmann Machines based on archetype selection

In the previous chapter, we described the classical implementation of a Restricted Boltzmann Machine (RBM). As mentioned in the introduction, these types of neural networks excel at learning probability distributions, which can be useful for solving probability density estimation and classification problems.

In this chapter we will introduce an alternative training method proposed in [1], based in archetype selection, that we will denote as *archetype* RBM (ARCH RBM). But first, let us define what an archetype is in the sense of learning algorithms.

Definition 2.1 *An **Archetype** or **Pattern** is a representative example or prototype of a group (cluster) of data points. They summarize and represent a large amount of data in a more compact and interpretable form.*

We will denote an archetype in our training set as $\boldsymbol{\xi}$.

Definition 2.2 *A **Blurred Example** is a variation or perturbation of the original archetype. These variations can be intentional, such as adding noise or blurring to the data, to simulate real-world conditions where the data may not be perfectly clean or clear.*

Blurred examples of an archetype are denoted as $\boldsymbol{\eta}^a$, where $a = 1, \ldots, M$, M being the total number of examples for that archetype. They can be used to augment the

training set and make the model able to recognize the underlying patterns that are invariant to the noise introduced in the data.

In the case of the ARCH RBM, these blurred examples are the only vectors seen by the network, while archetypes are the underlying patterns to be learnt despite never being seen. For intuition guidance, consider the MNIST dataset of handwritten digits [26]. We can interpret a certain pattern to be learnt ($\boldsymbol{\xi}$) as the archetype of the digit 7, while the set of examples that we provide our RBM with ($\{\boldsymbol{\eta}^a\}_{a=1,\dots,M}$) would correspond to the set of pictures of handwritten 7's.

2.1 Grandmother cell scenario

The term *grandmother cell* describes a theoretical neuron that represents a complex concept by its own [27]. This neuron activates when it receives the information about the specific entity it represents.

The key concept that can be applied to neural networks is that, in a *grandmother cell scenario*, the RBM is built of two layers:

- A visible layer with N binary neurons $\sigma_i = \pm 1, \quad i \in \{1, \dots, N\}$.
- The hidden layer, where we allocate **one neuron per archetype** or pattern. The neurons are also binary $z_\mu = \pm 1, \quad \mu \in \{1, \dots, K\}$. Where K is the number of different archetypes and therefore, the number of hidden neurons.

Definition 2.3 *Consider a set of $\{\boldsymbol{\xi}^\mu\}_{\mu=1,\dots,K}$ K archetypes. Then, the **Training Set** for the ARCH RBM is $\mathcal{S} = \{\boldsymbol{\eta}^{\mu,a}, \boldsymbol{z}^\mu\}_{\mu=1,\dots,K}^{a=1,\dots,M_\mu}$, where M_μ is the number of examples for archetype $\boldsymbol{\xi}^\mu$. Both $\boldsymbol{\xi}^\mu$ and $\boldsymbol{\eta}^{\mu,a}$ are encoded as binary vectors of length N, with values $\in \{-1, 1\}$; while $\boldsymbol{z}^\mu$ is the corresponding class label of the μ-th archetype $\boldsymbol{\xi}^\mu$ (and its subset $\{\boldsymbol{\eta}^{\mu,a}\}_{a=1,\dots,M_\mu}$) encoded as a vector of length K in the hidden layer in a one-hot fashion (i.e. $z_\mu^\mu = 1$ and $z_{\nu\neq\mu}^\mu = -1$).*

Note that we are using the notation from [1], this is convenient in order to differentiate this neural network from an SRBM, as both are designed for supervised learning.

Between both layers, there are connections represented by the weight matrix

$\boldsymbol{W} \in \mathbb{R}^{N\times K}$, with elements w_i^μ. As opposed to the previous case, there are no biases, so the energy function becomes

$$E(\boldsymbol{\sigma}, \boldsymbol{z}|\boldsymbol{W}) = -\beta \sum_{\mu=1}^{K}\sum_{i=1}^{N} w_i^\mu \sigma_i z_\mu. \tag{2.1}$$

Where β is introduced as a factor that ensures the linear scaling of the energy with respect to the size of the network. In our case, we will treat it as another parameter to help adjust our results.

The equilibrium distribution is still given by the Boltzmann-Gibbs distribution

$$\mathcal{P}(\boldsymbol{\sigma}, \boldsymbol{z}|\boldsymbol{W}) = \frac{1}{Z} e^{-E(\boldsymbol{\sigma},\boldsymbol{z})}. \tag{2.2}$$

However, since this RBM does not have bias (i.e., $\boldsymbol{b} = 0, \boldsymbol{c} = 0$), the marginal probability $p(\boldsymbol{\sigma})$ becomes

$$\begin{aligned} p(\boldsymbol{\sigma}) &= \sum_{\boldsymbol{z}} p(\boldsymbol{\sigma}, \boldsymbol{z}) = \frac{1}{Z}\sum_{\boldsymbol{z}} e^{-E(\boldsymbol{\sigma},\boldsymbol{z})} = \frac{1}{Z}\sum_{z_1}\cdots\sum_{z_K}\prod_{j=1}^{K} e^{\beta(\boldsymbol{\sigma}^T \boldsymbol{W}_j z_j)} \\ &= \frac{1}{Z}\prod_{j=1}^{K}\sum_{z_j} e^{\beta(\boldsymbol{\sigma}^T \boldsymbol{W}_j z_j)} = \frac{1}{Z}\prod_{j=1}^{K}(e^{-\beta(\boldsymbol{\sigma}^T \boldsymbol{W}_j)} + e^{\beta(\boldsymbol{\sigma}^T \boldsymbol{W}_j)}), \end{aligned} \tag{2.3}$$

On the other hand, the expression for the conditional probabilities become

$$\begin{aligned} p(\sigma_i|\boldsymbol{z}) &= \frac{1}{2}(1 + \tanh(\sum_{j=1}^{K}\beta(w_{ij} z_j))), \\ p(z_j|\boldsymbol{\sigma}) &= \frac{1}{2}(1 + \tanh(\sum_{i=1}^{N}\beta(w_{ij} \sigma_i))). \end{aligned} \tag{2.4}$$

The procedure is the same as in eq.1.6 and eq.1.7, we just need to change the notation and the expression inside the tanh.

2.2 Training of the ARCH RBM

The main difference between the SRBM and this archetype based network lies in its training process. The innovative method proposed in [1] consists in; instead

of letting the hidden layer evolve freely during the learning phase, we clamp both layers, namely setting the visible layer $\boldsymbol{\sigma}$ to one of the examples in the training set $\boldsymbol{\eta}^{\mu,a}$, while the hidden layer $\boldsymbol{z}$ is set to the corresponding label $\boldsymbol{z}^{(\mu)}$, as seen in fig.3. This also means that the number of hidden units N_h is fixed to be equal to the number of archetypes K.

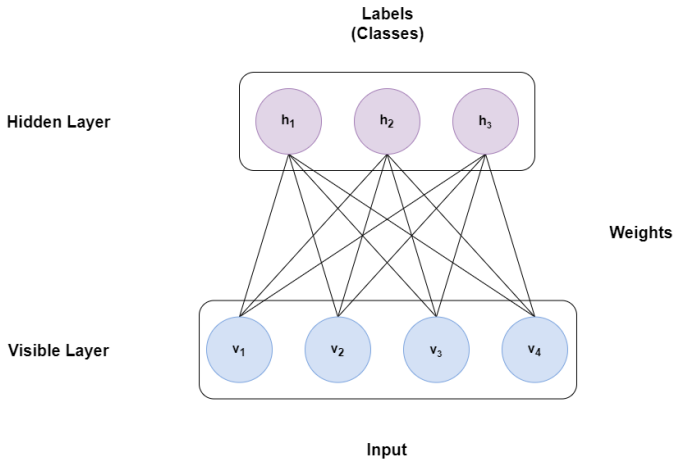


Figure 3: Diagram of an ARCH RBM with 3 encoded archetypes (and hidden units).

This training process can be seen as supervised learning, where the machine is trained using labeled data in order to classify or predict new examples. The grandmother-cell setting can be interpreted as the correspondence between the activation of each hidden neuron with the archetype they represent, forcing only one hidden neuron to be active.

We can compare the calculation of the gradients between the SRBM and the ARCH RBM. The latter is calculated according to

$$\Delta w_i^{\mu} \propto \left(\langle \sigma_i z_{\mu} \rangle_{clamped} - \langle \sigma_i z_{\mu} \rangle_{free} \right), \forall (i, \mu) \in (N \times K). \tag{2.5}$$

In eq.2.5, the clamped average corresponds to the positive phase of the SRBM in eq.1.15, and it is performed by setting $\boldsymbol{\sigma} = \boldsymbol{\eta}^{\mu,a}$ (where $\boldsymbol{\eta}^{\mu,a}$ corresponds to the a-th example of the digit μ) and $\boldsymbol{z} = \boldsymbol{z}^{\mu}$ and averaging their product over the batch of size b

$$\langle \boldsymbol{\sigma}^T \boldsymbol{z} \rangle_{clamped} = \frac{1}{b} \sum_{\boldsymbol{\eta}^{\mu,a} \in batch} (\boldsymbol{\eta}^{\mu,a})^T \boldsymbol{z}^{\mu}. \tag{2.6}$$

The free average, which would correspond to the negative phase of the SRBM, is approximated by initializing both layers with the same configurations as in the positive phase, and then performing a single step of Gibbs sampling. What we mean by that is to sample $\boldsymbol{\sigma}^{(t+1)} \sim p(\boldsymbol{\sigma}|\boldsymbol{z}^{(t)})$ and $\boldsymbol{z}^{(t+1)} \sim p(\boldsymbol{z}|\boldsymbol{\sigma}^{(t)})$ at the same time, as seen in Algorithm 2, instead of what we did in Algorithm 1. The expression then becomes:

$$\langle \boldsymbol{\sigma}^T \boldsymbol{z} \rangle_{free} = \frac{1}{b} \sum_{\boldsymbol{\eta}^{\mu,a} \in batch} (\boldsymbol{\eta}_{\mu,a}^{(k)})^T \boldsymbol{z}_{\mu}^{(k)}. \tag{2.7}$$

Note that we have vectorized the expressions in eq.2.6 and eq.2.7

Algorithm 2 Archetype contrastive divergence

Input: RBM $(\boldsymbol{\sigma}, \boldsymbol{z})$, training batch $\mathcal{S}$.
Output: gradient approximation $\Delta \boldsymbol{W}$.

```
init ΔW = 0_{N,K}.
for all the η^{μ,a}, z^μ ∈ S do
    η^{(0)}_{μ,a} ← η^{μ,a}
    z^{(0)}_μ ← z^μ
    for t = 0, ..., k − 1 do
        sample η^{(t+1)}_{μ,a} ~ p(σ|z^{(t)}_μ)
        sample z^{(t+1)}_μ ~ p(z|η^{(t)}_{μ,a})
    ΔW ← ΔW + (η^{μ,a})^T z^μ − (η^{(k)}_{μ,a})^T z^{(k)}_μ
```

After the network is trained, it can be used to classify, when providing a $\boldsymbol{\sigma}$ configuration and obtaining the $\boldsymbol{z}$ activations; or as a conditional generative model, by feeding the network a class encoder $\boldsymbol{z}^{\mu}$ and recovering an input vector of the corresponding archetype.

To summarize the differences between the SRBM implementation and this archetype-based RBM from [1], we list them as follows:

Supervised Restricted Boltzmann Machine (SRBM):

- **Topology**: Labels are encoded in the visible layer. Uses parameters $\boldsymbol{\theta} = \{\boldsymbol{b}, \boldsymbol{c}, \boldsymbol{W}\}$.

- **Training set**: Only has visible inputs. Each training example has its corresponding label appended to it $(\boldsymbol{\eta}^{\mu,a}, \boldsymbol{z}^{\mu})$.

- **Learning**: The hidden layer may evolve freely during the training process. Gradients are calculated according to alg.1.

- **Conditional probabilities**: The probability of a visible state given a label and vice versa are calculated as marginal probabilities (see eq.1.20 and eq.1.21).

Archetype Restricted Boltzmann Machine (ARCH RBM):

- **Topology**: Labels are encoded in the hidden layer (which has as many units as there are archetypes). Only uses $\boldsymbol{W}$ as a parameter.

- **Training set**: Contains both visible inputs and hidden inputs. Each training example $\boldsymbol{\eta}^{\mu,a}$ has its corresponding label $\boldsymbol{z}^{\mu}$.

- **Learning**: The hidden layer is clamped to the labels of each training example. The gradient is calculated according to alg.2.

- **Conditional probabilities**: The conditional probabilities are directly calculated as seen in eq.2.4.

From this point onwards, we discuss the different experiments we have performed once the ARCH RBM is trained, in order to measure its performance in both modes (classifier and conditional generator). The last part of this project will compare the results obtained in the next chapter with an SRBM using the same experiments as measurements.

Chapter 3

Results obtained for the ARCH RBM

The aim of this chapter is to measure how well the neural network that we implemented performs both as a classifier and as a generator. In addition, we want to reproduce the results of Ref.[1]; that is, to show numerically that there exists a threshold size M_x, depending on r, such that the RBM correctly learns the archetypes.

However, without knowing specifically how they implemented the ARCH RBM, nor having the exact combination of parameters used in each evaluation, we can only test our implementation by trying to replicate their measuring procedures and comparing the results. Before proceeding with the experiments, we first define the dataset that is used on all experiments, as well as the measurements that will be taken in each case.

Definition 2.3 provides a general definition of the training set for the ARCH RBM. Here, we restrict to the case where the number of examples per archetype is the same for all archetypes, that is, $M_\mu = M$ for $\mu = 1, \dots, K$. In addition, we define archetype entries $\{\boldsymbol{\xi}^\mu\}_{\mu=1,\dots,K}$ as binary random variables drawn with probability

$$\mathcal{P}(\xi_i^\mu = +1) = \mathcal{P}(\xi_i^\mu = -1) = 1/2, \quad i = 1, \dots, N. \tag{3.1}$$

while the training examples are built applying a random distortion to the archetypes,

for any given $\mu = 1, \ldots, K$ and $a = 1, \ldots, M$, such that

$$\eta_i^{\mu,a} = \xi_i^\mu \chi_i^{\mu,a}, \quad i = 1, \ldots, N, \tag{3.2}$$

with $\mathcal{P}(\chi_i^{\mu,a} = +1) = 1 - \mathcal{P}(\chi_i^{\mu,a} = -1) = p \in [1/2, 1]$.

In this way, the closer p gets to $1/2$, the larger the distortion applied to the original pattern is. Thus, each value of μ defines a subset $\{\boldsymbol{\eta}^{\mu,a}\}_{a=1,\ldots,M}$ of noisy versions of $\boldsymbol{\xi}^\mu$. Likewise, each training example $\boldsymbol{\eta}^{\mu,a}$ and archetype $\boldsymbol{\xi}^\mu$ has a label or class that is encoded as a one-hot vector of length K in the hidden layer, denoted as $\boldsymbol{z}^\mu$, with $z_\mu^\mu = 1$ and $z_{\nu\neq\mu}^\mu = -1$.

Definition 3.1 *The **dataset quality**, denoted as r, is a metric that quantifies the degree of distortion applied to the training set. It is defined as*

$$r := 2p - 1 \in [0, 1],$$

where p is the parameter that determines $\mathcal{P}(\chi_i^{\mu,a} = +1)$.

Hence, when $r = 0$, it indicates that $p = 1/2$ and therefore, there is no correlation between examples and archetypes. Conversely, when $r = 1$, it implies that no distortion has been introduced, and each example perfectly matches its corresponding archetype pattern. In summary, the lower r is, the larger the distortion applied to the archetype becomes.

Now, we define the measurements taken in the experiments.

Definition 3.2 *The **classification probabilities** correspond to the probability of reaching a correct hidden configuration $\boldsymbol{z}^\mu$ when feeding the RBM with either an archetype or one of its distorted members of the training set*

$$\mathcal{P}(\boldsymbol{z}^\mu | \boldsymbol{\sigma} = \boldsymbol{\xi}^\mu) = \frac{1}{2}\left(1 + \tanh\left(\beta(\boldsymbol{\xi}^\mu)^T \boldsymbol{W}\right)\right) \tag{3.3}$$

and

$$\mathcal{P}(\boldsymbol{z}^\mu | \boldsymbol{\sigma} = \boldsymbol{\eta}^{\mu,a}) = \frac{1}{2}\left(1 + \tanh\left(\beta(\boldsymbol{\eta}^{\mu,a})^T \boldsymbol{W}\right)\right) \tag{3.4}$$

These expressions correspond to the conditional probabilities in eq.2.4, for $\boldsymbol{z} = \boldsymbol{z}^\mu$ and $\boldsymbol{\sigma} = \boldsymbol{\xi}^\mu, \boldsymbol{\eta}^{\mu,a}$ respectively.

Definition 3.3 *The **overlap** between a visible neuron configuration $\boldsymbol{\sigma}$ and the archetype $\boldsymbol{\xi}^\mu$ is defined as*

$$m^\mu := \frac{1}{N}\sum_{i=1}^{N} \xi_i^\mu \sigma_i, \quad \mu = 1, \dots, K, \tag{3.5}$$

while the overlap between $\boldsymbol{\sigma}$ and $\boldsymbol{\eta}^{\mu,a}$, for $a = 1, \dots, M$ and $\mu = 1, \dots, K$, is

$$n^{\mu,a} := \frac{1}{N}\sum_{i=1}^{N} \eta_i^{\mu,a} \sigma_i. \tag{3.6}$$

*Then, the **average overlap** over all archetypes is defined as*

$$\langle m \rangle = \frac{1}{K}\sum_{\mu=1}^{K} m^\mu, \tag{3.7}$$

while the average overlap over all examples is

$$\langle n \rangle = \frac{1}{KM}\sum_{\mu,a=1}^{K,M} n^{\mu,a}. \tag{3.8}$$

Lastly, we need to define the threshold size M_x that we expect to estimate with our experiments.

Definition 3.4 *The **threshold size** M_x corresponds to the minimum number of examples per archetype in the training set M such that the archetype measurements prevail over the example ones. That is, $\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma} = \boldsymbol{\xi}^\mu) > \mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma} = \boldsymbol{\eta}^{\mu,a})$ and $\langle m \rangle > \langle n \rangle$.*

3.1 Performance as a classifier

First, we test our trained RBM according to its ability to infer the archetype class associated to each input vector. In order to do that, we compute the probability of reaching a correct hidden configuration $\boldsymbol{z}^\mu$ when feeding the machine with either an archetype $\boldsymbol{\xi}^\mu$ or one of its distorted members of its subset $\{\boldsymbol{\eta}^\mu\}_{a=1,\dots,M}$, as in eq.3.3 and eq.3.4.

However, as we mentioned at the beginning of the chapter, we want to find the parameters that yield similar values to the ones shown in [1]; therefore, in the first experiment we focus on exploring the effects of varying certain parameters on the classification probabilities during the learning phase.

3.1.1 Classification probabilities over the training process

For this experiment, we start with a simple problem corresponding to an RBM of $N = 10$ visible neurons and $K = 5$ archetypes (and hidden neurons). In addition, we set all our training hyperparameters to the recommended values according to [18] (i.e., $\epsilon = 0.001, \lambda = 10^{-6}, \beta = 1.0$), and train the RBM with exact copies of the archetypes (i.e., $r = 1.0$ and $M = 1$). During the training process, we compute both the KL divergence and the classification probabilities, so as to compare the results after repeating the experiment for different values of the parameters.

It is worth mentioning that the KL divergence is calculated from the joint probability of the training set examples with their corresponding class. In this way, just like a standard KL calculation, it is used as a measure of the training success as well as taking into account the correct pairing of the examples with their labels. Its expression is as follows:

$$\begin{aligned} KL_{\sigma,z} &= \sum_{\boldsymbol{\eta}^{\mu,a} \in \mathcal{S}} q(\boldsymbol{\eta}^{\mu,a}, \boldsymbol{z}^{\mu}) \ln q(\boldsymbol{\eta}^{\mu,a}, \boldsymbol{z}^{\mu}) - \sum_{\boldsymbol{\eta}^{\mu,a} \in \mathcal{S}} q(\boldsymbol{\eta}^{\mu,a}, \boldsymbol{z}^{\mu}) \ln p(\boldsymbol{\eta}^{\mu,a}, \boldsymbol{z}^{\mu}) \\ &= \ln\left(\frac{1}{KM}\right) - \frac{1}{KM} \sum_{\boldsymbol{\eta}^{\mu,a} \in \mathcal{S}} \ln p(\boldsymbol{\eta}^{\mu,a}, \boldsymbol{z}^{\mu}), \end{aligned} \tag{3.9}$$

where we assume that $q(\boldsymbol{\eta}^{\mu,a}, \boldsymbol{z}^{\mu}) = \frac{1}{KM}$ for $\mu = 1, \ldots, K$ and $a = 1, \ldots, M$, KM being the size of the training set.

Table 1 shows the results of the experiment along with observations on the effect of varying different training parameters. Note that there are two columns for the initial and final values of the KL divergence, $KL^i_{\sigma,z}$ and $KL^f_{\sigma,z}$, respectively.

With the first set of parameters, we train the RBM using a single ($M = 1$) exact copy of each archetype. Then, classifying an archetype and an example is the same and they both tend to 1.0 at the end of the learning process, as expected.

Parameters							Results		
N	K	r	M	ϵ	λ	β	$KL^i_{\sigma,z}$	$KL^f_{\sigma,z}$	Observation
10	5	1.0	1	10^{-3}	10^{-6}	1.0	8.791	1.616	Both probabilities tend to 1.0, as expected.
25	5	1.0	1	10^{-3}	10^{-6}	1.0	19.260	1.148	Faster convergence to the same results.
25	5	0.8	2	10^{-4}	10^{-6}	1.0	18.413	6.148	Worse final KL, probabilities still tend to 1.0.
25	5	0.8	8	10^{-4}	10^{-6}	1.0	17.093	8.623	Lower $\mathcal{P}(\boldsymbol{z}^\mu\|\boldsymbol{\eta}^{\mu,a})$ value, $\mathcal{P}(\boldsymbol{z}^\mu\|\boldsymbol{\xi}^\mu)$ does not change.
50	5	0.8	8	10^{-4}	10^{-6}	1.0	34.352	19.356	Overflow error, probabilities are saturated at 1.0.
50	5	0.8	8	10^{-3}	10^{-6}	$1/\sqrt{N}$	34.432	20.355	Scaling down energy is not enough to avoid overflow.
50	5	0.8	8	10^{-3}	10^{-4}	$1/\sqrt{N}$	34.428	20.198	Increasing λ is also not enough.
50	5	0.8	8	10^{-1}	10^{-4}	$1/N$	34.435	23.285	Scaling down energy further worked.
50	5	0.6	8	10^{-1}	10^{-4}	$1/N$	34.433	26.022	**Values resemble their results for $M > M_x$ case.**
50	5	0.6	2	1.0	10^{-4}	$1/N$	36.512	20.160	Values are not similar to the ones in Figure 1 in [1].
50	5	0.4	3	1.0	10^{-4}	$1/N$	35.415	27.549	**Values resemble their results for $M < M_x$ case.**

Table 1: Parameters used for the measurement of the probabilities and their respective results.

Overall, as the number of visible neurons grows, so does the initial KL Divergence, which is not surprising, since the only term in eq.3.9 that is not constant is $p(\boldsymbol{\eta}^{\mu,a}, \boldsymbol{z}^\mu)$ and at the beginning of the learning process it is close to $1/2^N$ (i.e., all states have the same uniform probability before training the network). On the other hand, decreasing the value of r increases the distortion in our training examples, causing the final values of the KL and the conditional probabilities to worsen. However, if we increase the number of examples per archetype M for the same value of r, the learning improves as well as the resulting KL and $\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma} = \boldsymbol{\xi}^\mu)$ and $\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma} = \boldsymbol{\xi}^\mu)$.

We have observed that as we increase N further, the classification conditional probabilities saturate to 1.0, making it difficult to interpret the results of the experiment. To address this issue, we set the parameter $\beta = 1/N$ (see eq.2.1) to ensure that the hyperbolic tangent used to calculate these probabilities in eq.3.3-3.4 scales properly with N.

This has allowed us to obtain more accurate results and to find two different sets of parameters that produce curves similar to those found in Figure 1 in [1].

From fig.4, we can see the difference between the two cases. The case with a larger M seems to classify archetypes better than examples (i.e., $p(\boldsymbol{z}^\mu|\boldsymbol{\xi}^\mu) > p(\boldsymbol{z}^\mu|\boldsymbol{\eta}^{\mu,a})$), whilst having a relatively small number of examples makes the system prioritize the distorted examples instead ($p(\boldsymbol{z}^\mu|\boldsymbol{\xi}^\mu) < p(\boldsymbol{z}^\mu|\boldsymbol{\eta}^{\mu,a})$).

In the next experiment, we check whether a critical size M_x exists for which the ARCH RBM starts to extract and learn the archetype from the blurred samples. This is one of the most important aspects of this work, as it showcasts and measures explicitely the generalizing capabilities of the network.

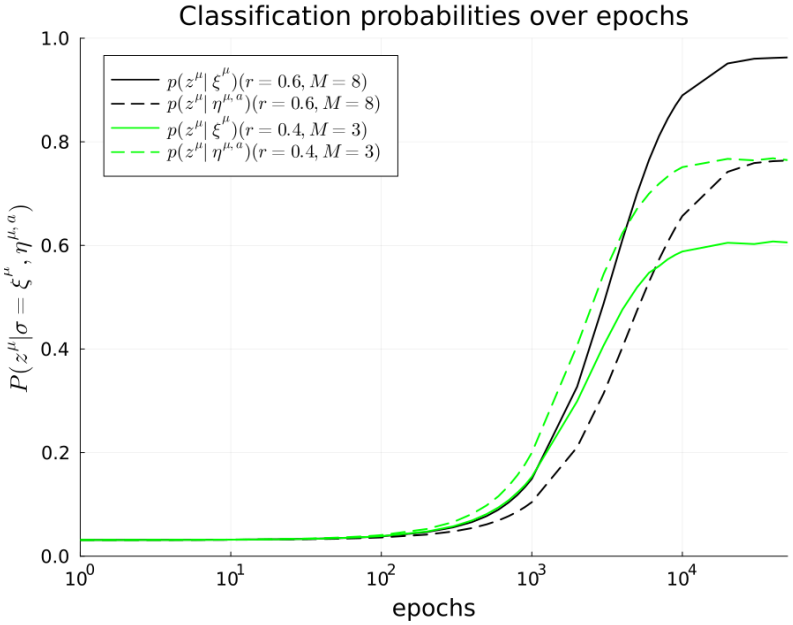


Figure 4: Evolution of the classification probabilities over the epochs. Black curves parameters: $N = 50$, $K = 5$, $M = 8$, $r = 0.6$, $\epsilon = 10^{-1}$, $\lambda = 10^{-4}$, $\beta = 1/N$. Green curves parameters: $N = 50$, $K = 5$, $M = 3$, $r = 0.4$, $\epsilon = 1.0$, $\lambda = 10^{-4}$, $\beta = 1/N$.

3.1.2 Crossover size M_x for the classification probabilities

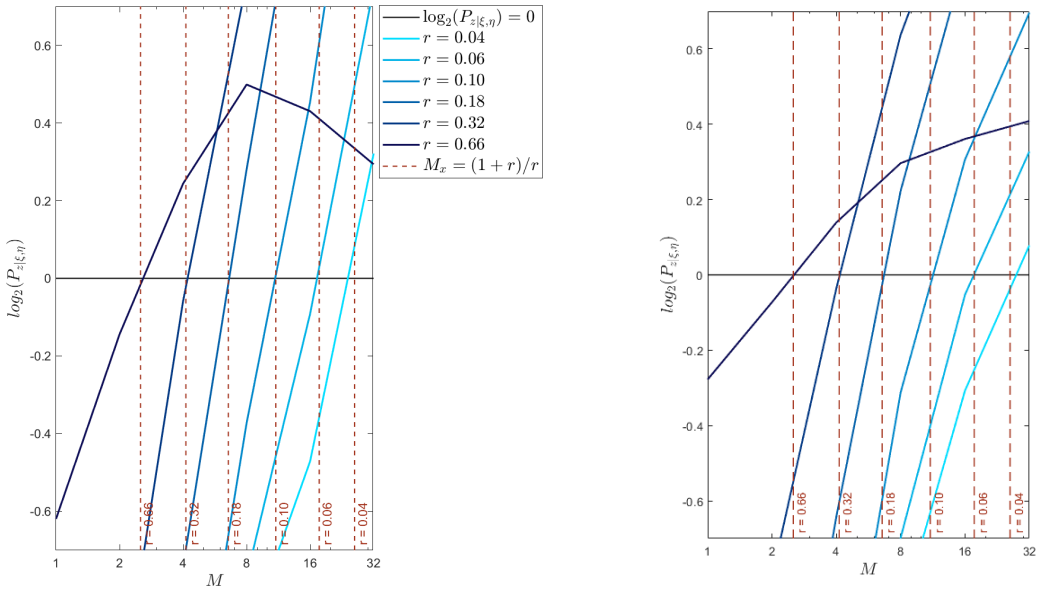
For this experiment, we are interested in calculating the ratio of the classification probabilities for different values of r and M, according to the expression

$$\mathcal{P}_{\boldsymbol{z}|\boldsymbol{\xi},\boldsymbol{\eta}} := \frac{1}{KM}\sum_{\mu,a=1}^{K,M}\frac{\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma}=\boldsymbol{\xi}^\mu)}{\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma}=\boldsymbol{\eta}^{\mu,a})}. \tag{3.10}$$

In this way, if the ARCH RBM classifies the archetypes better than the examples, all the ratios would be > 1 and therefore the average too. On the contrary, if it classifies the distorted examples better, the ratio would be < 1.

We train the RBM using the set of parameters found in the previous experiment (i.e., $N = 50$, $K = 5$, $\epsilon = 10^{-1}$, $\lambda = 10^{-4}$, $\beta = 1/N$) and a starting value of $r = 0.66$ and $M = 1$. After the training has ended, we calculate the ratio in eq.3.10 and repeat the experiment for each value of M in the range $M = 1, \ldots, 32$. In this way, we can examine how the classification performance changes as we provide the network with more examples. We repeat this procedure for each $r \in \{0.66, 0.32, 0.18, 0.10, 0.06, 0.04\}$. By doing so, we also determine whether there is any relation between the threshold size M_x and the quality of the

examples r.



(a) Ratio of probabilities with $N = 50$.

(b) Ratio of probabilities with $N = 200$.

Figure 5: Performance of the RBM as a classifier, showing the existence of the crossover size M_x as the crossing points with the horizontal line. Parameters used: $K = 5$, $\epsilon = 10^{-1}$, $\lambda = 10^{-4}$, $\beta = 1/N$.

Figure 5 shows the logarithm of the ratio between the conditional probabilities versus the number of examples per archetype M, in logarithmic scale, for different choices of the dataset quality r.

According to Definition 3.4, the threshold size M_x corresponds to the number of examples for which $\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma} = \boldsymbol{\xi}^\mu) = \mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma} = \boldsymbol{\eta}^{\mu,a})$, or in other words, when the ratio $\mathcal{P}_{\boldsymbol{z}|\boldsymbol{\xi},\boldsymbol{\eta}} = 1.0$. In fig.5a, this can be seen as the crossing points of the curves with the horizontal line ($\log_2(\mathcal{P}_{\boldsymbol{z}|\boldsymbol{\xi},\boldsymbol{\eta}}) = 0$) and, as one can observe, the value of M_x is related to the dataset quality r. Furthermore, we repeated the experiment with a different number of visible neurons $N = 200$ and plotted the results in fig.5b so as to verify that M_x does not depend on N.

We can conclude this section by affirming that there exists a threshold size M_x for which our network generalizes the concept of the archetype hidden within the blurred examples. Furthermore, we can see a certain relation between this critical size and the quality of those examples, which is found in [1] to follow the expression $M_x = (1+r)/r$. For instance, we can check that this expression gives us the crossing

points in fig.5: if we take $r = 0.66$, then $M_x = (1+0.66)/0.66 \simeq 2.52$, which is where the darkest curve in the figure intersects the horizontal line; another example would be for $r = 0.04$, its corresponding $M_x = (1 + 0.04)/0.04 = 26$, which is roughly the crossing point of the brightest curve in the figure.
In the next section we analyze how the RBM performs as a generative model to see whether this critical size also exists in this mode.

3.2 Performance as a conditional generator

Next, we can test the RBM as a generative model using a different procedure. After the network has finished the training, we feed it a $\boldsymbol{z}^\mu$ configuration and calculate the expected state of the visible neurons given that class, that we will denote as $\hat{\boldsymbol{\sigma}}^\mu$. The i-th element of this expected state can be calculated analytically with the following expression, for $i = 1, \dots, N$:

$$\begin{aligned}\mathbb{E}_{p(\boldsymbol{\sigma}|\boldsymbol{z}^\mu)}\left[\sigma_i\right] &= \sum_{\boldsymbol{\sigma}} \sigma_i p(\boldsymbol{\sigma}|\boldsymbol{z}^\mu) = \underbrace{\sum_{\sigma_1} p(\sigma_1|\boldsymbol{z}^\mu)}_{=1} \cdots \sum_{\sigma_i} \sigma_i p(\sigma_i|\boldsymbol{z}^\mu) \cdots \underbrace{\sum_{\sigma_N} p(\sigma_N|\boldsymbol{z}^\mu)}_{=1} \\ &= \sum_{\sigma_i \in \{-1,1\}} \sigma_i p(\sigma_i|\boldsymbol{z}^\mu) = 1 \cdot p(\sigma_i = 1|\boldsymbol{z}^\mu) - 1 \cdot p(\sigma_i = -1|\boldsymbol{z}^\mu) \\ &= 2 \cdot p(\sigma_i = 1|\boldsymbol{z}^\mu) - 1 = \tanh\left(\beta \sum_{j=1}^{K} w_{ij} z_j^\mu\right), \end{aligned} \tag{3.11}$$

which can be vectorized as $\hat{\boldsymbol{\sigma}}^\mu = \mathbb{E}_{p(\boldsymbol{\sigma}|\boldsymbol{z}^\mu)}\left[\boldsymbol{\sigma}\right] = \tanh(\beta \boldsymbol{W} \boldsymbol{z}^\mu)$.

In this way, we can measure the overlap (see Definition 3.3) between the obtained visible neuron configuration $\hat{\boldsymbol{\sigma}}^\mu$ and $\boldsymbol{\xi}^\mu$ and compare it with the overlap between $\hat{\boldsymbol{\sigma}}^\mu$ and $\boldsymbol{\eta}^{\mu,a}$, according to eq.3.5-3.6. Then, we average them over the different choices of hidden states (i.e., $\mu \in [1, K]$), and over the number of examples per archetype M as seen in eq.3.7-3.8.

As in the previous case, we first need to find the set of parameters with which we obtain similar values as in [1]. This first experiment focuses on varying the parameters and looking how it affects the overlaps $\langle m \rangle$ and $\langle n \rangle$ during the training process.

3.2.1 Overlaps $\langle m \rangle$ and $\langle n \rangle$ over the training process

Starting with a simple RBM of $N = 10$ visible neurons and $K = 5$ hidden neurons (1 per archetype), trained with exact copies of the archetypes using $r = 1.0$ and $M = 1$ as well as the default values of the training hyperparameters, we compute the KL error measure and the overlaps. Then, we keep repeating the procedure while changing the value of one parameter each time.

Parameters							Results		
N	K	r	M	ϵ	λ	β	$KL^i_{\sigma,z}$	$KL^f_{\sigma,z}$	Observation
10	5	1.0	1	10^{-3}	10^{-6}	1.0	8.791	1.616	Both $\langle m \rangle$ and $\langle n \rangle$ tend to 1.0.
25	5	1.0	1	10^{-3}	10^{-6}	1.0	19.136	3.421	It converges faster, but with more error.
25	5	0.8	2	10^{-4}	10^{-6}	1.0	18.094	7.754	Lower r yields worse values of KL and overlaps.
25	5	0.8	8	10^{-4}	10^{-6}	1.0	17.093	8.623	Faster convergence, better $\langle n \rangle$ value, $\langle m \rangle$ is unchanged.
25	2	0.8	8	10^{-3}	10^{-6}	1.0	15.923	12.029	Faster convergence with more error. $\langle m \rangle$ and $\langle n \rangle$ have lower values.
25	10	0.8	8	10^{-5}	10^{-6}	1.0	19.891	13.789	Increased computation time. $\langle m \rangle$ and $\langle n \rangle$ did not converge after training.
50	5	0.8	8	10^{-4}	10^{-6}	1.0	34.352	19.356	Both overlaps now have similar values to the ones in figure 1 in [1].
50	5	0.9	8	10^{-4}	10^{-6}	1.0	34.348	11.558	$\langle m \rangle$ value is correct, $\langle n \rangle$ is different.
50	5	0.9	16	10^{-4}	10^{-6}	1.0	33.748	14.951	Convergence a bit faster. **Used for $M > M_x$ case**.
50	5	0.9	1	10^{-3}	10^{-6}	1.0	36.527	2.915	$\langle n \rangle$ is not correct.
50	5	0.6	1	10^{-3}	10^{-6}	1.0	36.461	2.343	Both curves resemble the source's figure. **Used for $M < M_x$ case**.

Table 2: Parameters used for the measurement of the overlaps and their respective results.

Table 2 summarizes the obtained results. At first, since the examples have no distortion, the expression for $\langle n \rangle$ is the same as $\langle m \rangle$ and they both tend to 1.0 at the end of the training phase, which is expected.

In short, increasing the number of visible neurons N lead to better values of both overlaps. In addition, we tried to increase the *load* of the network by reducing the number of hidden neurons (and number of archetypes K) with respect to N. This introduced more error and worse values for both $\langle m \rangle$ and $\langle n \rangle$.

On the other hand, since the calculation of the overlaps only involves dot products, it did not lead to saturated values. So, we did not need to adjust the value of β accordingly.

Figure 6 shows the same behaviour as in the evolution of the classification probabilities (see fig.4). When the number of examples provided to the network is large enough, the system prioritizes generating states closer to the archetypes than they are to examples.

Proceeding as before, in the next experiment we will check whether the critical size M_x exists for the generator RBM, and if its value is the same as in the

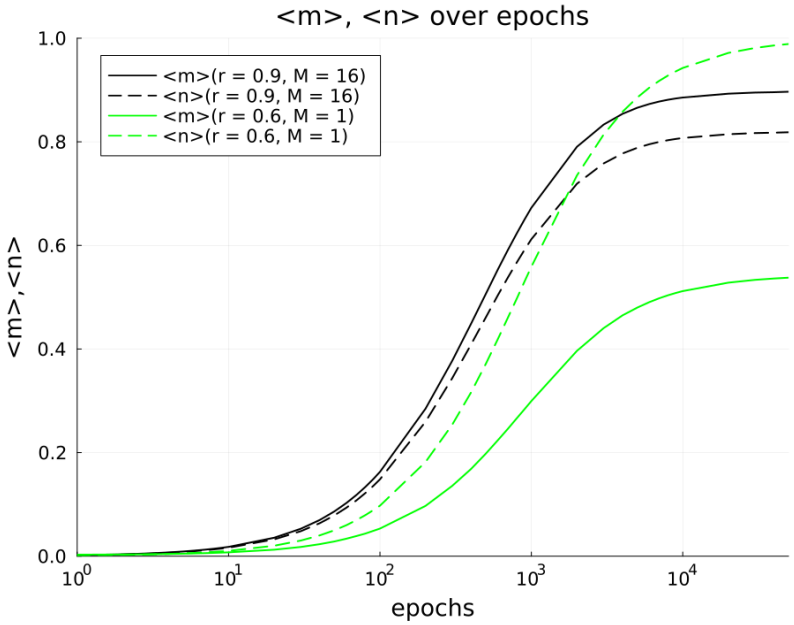


Figure 6: Evolution of the overlaps over the epochs. Black curves parameters: $N = 50$, $K = 5$, $M = 16$, $r = 0.9$, $\epsilon = 10^{-3}$, $\lambda = 10^{-6}$, $\beta = 1.0$. Green curves parameters: $N = 50$, $K = 5$, $M = 1$, $r = 0.6$, $\epsilon = 10^{-3}$, $\lambda = 10^{-6}$, $\beta = 1.0$.

classification mode.

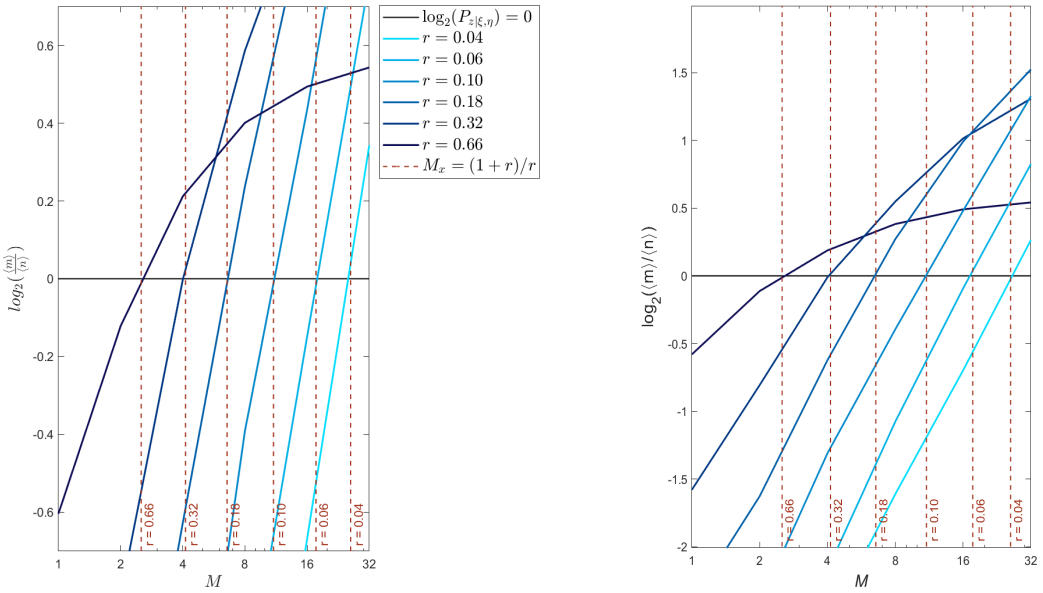
3.2.2 Crossover size M_x for the generative model

Similar to the previous case, we use the set of parameters that we have found ($N = 50$, $K = 5$, $\epsilon = 10^{-3}$, $\lambda = 10^{-6}$, $\beta = 1.0$), along with a starting value of $r = 0.66$ and $M = 1$, to train the RBM and calculate the ratio $\langle m \rangle / \langle n \rangle$. We repeat the experiment for different values of $M = 1, \ldots, 32$ to obtain a single curve corresponding to the performance of the RBM working as a generator when trained with examples of quality $r = 0.66$. Following the same procedure for each $r \in \{0.66, 0.32, 0.18, 0.10, 0.06, 0.04\}$ we obtain Figure 7.

In fig.7a we can see that the crossing points of the curves with the horizontal line ($\log_2(\langle m \rangle / \langle n \rangle) = 0$) coincide with the values of M_x for each r value that we found in fig.5. Once again, we repeated the experiment with $N = 200$ in order to confirm that the threshold size does not depend on the problem size.

Finally, we can confirm the existence of the threshold and its relation to the quality of the dataset. However, we still have one more parameter that we can study for its effect on the crossover size M_x and that is the load of the network.

In the final section of this chapter, we will estimate the critical size by finding the



(a) Ratio of $\langle m \rangle / \langle n \rangle$ with $N = 50$.

(b) Ratio of $\langle m \rangle / \langle n \rangle$ with $N = 200$.

Figure 7: Performance of the RBM as a generator, showing the existence of the crossover size M_x.

crossing points of the classification probability curves while varying the number of archetypes K.

3.3 Crossover size M_x with respect to the load α

In this section, we focus on checking whether there is any relation between the crossover size M_x and the load of the ARCH RBM, which is defined as

$$\alpha = \lim_{N \to \infty} \frac{K}{N}. \tag{3.12}$$

The setup for this experiment is similar to what we did in Section 3.1; we train the RBM and calculate the correct classification conditional probabilities for each value of M, then repeat for the r's and finally do the same for $K = \{2, 4, 8, 16, 32\}$. This way, in addition to visualizing the crossing points between $\mathcal{P}(\boldsymbol{z}^\mu | \boldsymbol{\sigma} = \boldsymbol{\xi}^\mu)$ and $\mathcal{P}(\boldsymbol{z}^\mu | \boldsymbol{\sigma} = \boldsymbol{\eta}^{\mu,a})$ by explicitly plotting the individual classification probabilities, we can observe the dependency of these crossing points with the number of archetypes K.

Figure 8, shows the classification probabilities ($\mathcal{P}(\boldsymbol{z}^\mu | \boldsymbol{\sigma} = \boldsymbol{\xi}^\mu)$ dark curves and $\mathcal{P}(\boldsymbol{z}^\mu | \boldsymbol{\sigma} = \boldsymbol{\eta}^{\mu,a})$ bright curves) for $K = 2$ (the other figures for the different K are

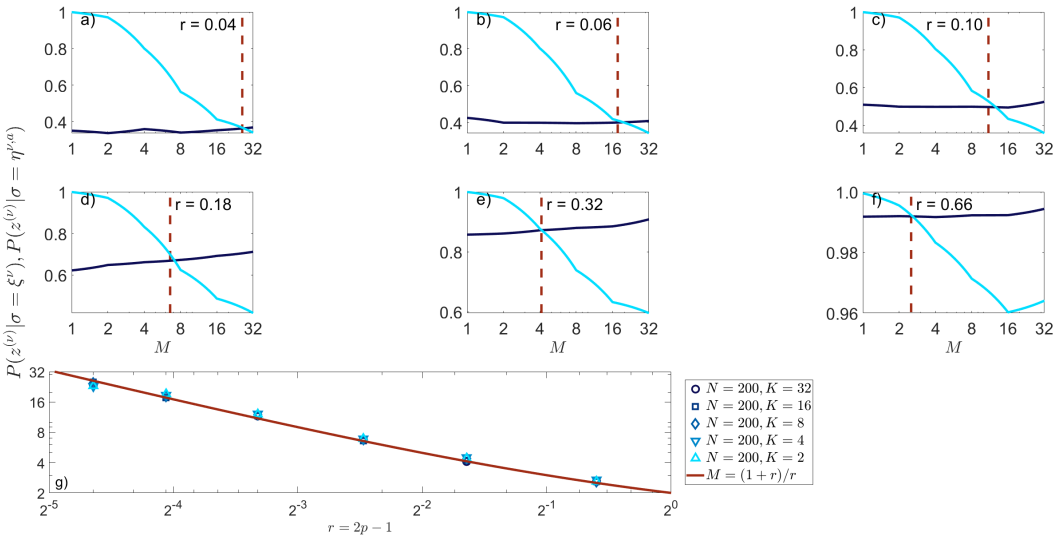


Figure 8: Numerical estimation of the crossover size M_x and its relation to α. Other fixed parameters are $N = 200, \epsilon = 10^{-1}, \lambda = 10^{-4}, \beta = 1/N$.

very similar) in panels $a - f$. The vertical dashed lines correspond to the values of M_x, and each panel corresponds to a particular value of the dataset quality, ranging from 0.04 to 0.66. In panel g, the solid line $M_x = (1 + r)/r$ divides two regions: the top right region is where the network is able to generalize and learn the concept of the archetype, whereas below it all examples are considered distinct patterns.

We can observe the curves in panels $a - f$ intersect exactly on the expected values (the vertical dashed lines). These crossing points correspond to the M_x values shown in panel g, depicted by the different markers for each load value. One can see that these markers follow the red line for all values of r, and do not depend on the load of the ARCH RBM, since they all have the same values for the crossing points.

We conclude this chapter by stating that we have managed to adequately replicate the ARCH RBM implementation and have obtained results that confirm the existence of a critical dataset size at which the neural network recognizes and learns the original archetype. Likewise, we have also found that this threshold M_x does **not** depend on the RBM load, but only on the quality of the examples r.

In the next chapter, we will compare the results of these experiments with a Supervised Restricted Boltzmann Machine (SRBM), provided with the same blurred examples.

Chapter 4

Comparative results obtained for the SRBM

In this chapter, we will repeat the same experiments for the SRBM and compare the results with what we obtained for the ARCH RBM. In order to test the SRBM as we did for the ARCH RBM, we use the same one-hot encoded labels $\{\boldsymbol{z}^{\mu}\}_{\mu=1,\dots,K}$, where K is the number of different archetypes. As we mentioned before, these labels are appended to their corresponding training set examples $\{\boldsymbol{\eta}^{\mu,a}\}_{a=1,\dots,M}$, resulting in a visible input $\left(\eta_1^{\mu,a},\dots,\eta_N^{\mu,a},z_1^{\mu},\dots,z_K^{\mu}\right)$. Thus, the total number of visible units is $N_v = N + K$, where N is the length of the vectors encoding both the archetypes and their distorted copies.

In this way, the hidden layer is not needed to pair the visible input with its class and it may evolve freely during the learning phase. Therefore, the number of the hidden units is treated as another parameter that can be tuned depending on the specific problem.

4.1 SRBM performance as a classifier

4.1.1 Threshold size M_x with the classification probabilities

For this experiment, we calculate the ratio of the classification probabilities defined in eq.1.20 as

$$\mathcal{P}^{SRBM}_{\boldsymbol{z}|\boldsymbol{\xi},\boldsymbol{\eta}} := \frac{1}{KM}\sum_{\mu,a=1}^{K,M}\frac{\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\xi}^\mu)}{\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\eta}^{\mu,a})}, \tag{4.1}$$

which we use as a measure of the performance of the SRBM as a classifier. In this way, when the ratio is > 1, the network classifies the underlying archetypes better than the training set examples and vice versa. Recall that the probabilities in eq.4.1 are calculated as marginal probabilities.

To start the experiment, we use the problem parameters $N = 200$, $K = 4$, $r = 0.66$ and $M = 1$. We train the SRBM with the same dataset as the ARCH RBM (see eq.3.1-3.2), while the rest of hyperparameters are found with hyperparameter tuning (Subsection 1.3.3). Then, we calculate the ratio shown in eq.4.1 and repeat the procedure for each $M = 1, \ldots, 32$ to obtain the classification performance curve for $r = 0.66$. We repeat the same process while varying the value of $r = \{0.66, 0.32, 0.18, 0.10, 0.06, 0.04\}$. The results are shown in Figure 9

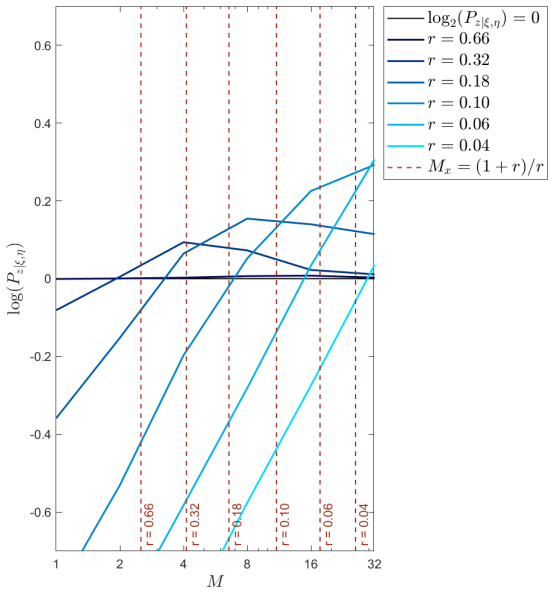


Figure 9: Performance of the SRBM as a classifier. Parameters used: $N = 200, N_h = 50, K = 4, \epsilon = 10^{-3}, \lambda = 10^{-6}$. Vertical dashed lines correspond to theoretical values of $M_x = (1 + r)/r$.

One can see that this figure is different from Figure 5, as the curves for all values of r do not cross the horizontal line at the theoretical M_x values (which are depicted in the figure as the vertical dashed lines). This may mean that, for the SRBM, the threshold value M_x depends on another parameter such as the load α.

Moreover, all curves seem to converge back to $\log_2(\mathcal{P}_{\boldsymbol{z}|\boldsymbol{\xi},\boldsymbol{\eta}}) = 0$ as M increases,

especially in the case of $r = 0.66$ which is almost constantly 0. A plausible explanation for this is that $\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\xi}^\mu)$ reaches 1 with a smaller M than the ARCH RBM, and increasing the number of examples also makes $\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\eta}^{\mu,a})$ tend to 1. Hence why the logarithm of the ratio of probabilities converges back to 0.

4.2 SRBM performance as a generative model

Next, we test the SRBM as a generative model by calculating the overlaps (see Definition 3.3) between the expected visible state $\tilde{\boldsymbol{v}}^\mu$ defined in eq.4.2 and both the underlying archetypes $\boldsymbol{\xi}^\mu$ and the distorted examples $\boldsymbol{\eta}^{\mu,a}$.

$$\hat{\boldsymbol{v}}^\mu = \mathbb{E}_{p(\boldsymbol{v}|\boldsymbol{z}^\mu)}[\boldsymbol{v}] = \sum_{\boldsymbol{v}} \boldsymbol{v} \cdot p(\boldsymbol{v}|\boldsymbol{z}^\mu), \tag{4.2}$$

where $p(\boldsymbol{v}|\boldsymbol{z}^\mu)$ is calculated as in eq.1.21 and is very expensive to compute ($\mathcal{O}(2^N)$). However, we can compute this expectation for a simple problem with $N = 20$ in order to test the performance of the SRBM and average the results for multiple realizations.

We start with $N = 20$, $K = 4$, $r = 0.66$ and $M = 1$; train the SRBM, calculate $\tilde{\boldsymbol{v}}^\mu$ and use it to find the overlaps m^μ and $n^{\mu,a}$, for each value of $\mu = 1, \ldots, K$ and $a = 1, \ldots, M$. Then, we average the results into $\langle m \rangle$ and $\langle n \rangle$ (see Definition 3.3), respectively. We repeat this procedure for each value of $M = 1, \ldots, 32$ and $r = \{0.66, \ldots, 0.04\}$, as we did in the previous experiment.

Figure 10 shows the performance of the SRBM as a generator. In this case, the curves are similar to the ones we obtained for the ARCH RBM (see fig.7). Nonetheless, the crossing points of these curves with the line $\log_2(\langle m \rangle / \langle n \rangle) = 0$ do not coincide with the theoretical values despite being very close in some cases.

In the last section of this chapter, we look directly at the classification probability curves calculated for different load values (i.e., varying K with respect to N), in order to see what is causing these differences between the two networks.

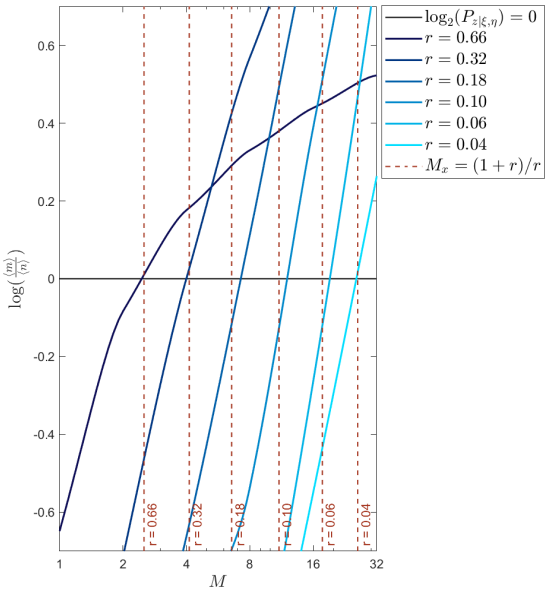


Figure 10: Performance of the SRBM as a generator. Parameters used: $N = 10, N_h = 50, K = 2, \epsilon = 10^{-3}, \lambda = 10^{-6}$.

4.3 Crossover size M_x with respect to the load α

To conduct this experiment, we follow the procedure used in Section 4.1.1 and repeat for different values of $K = \{2, 4, 8, 16, 32\}$.

Figure 11 shows in panels $a-f$ the probabilities of correctly classifying the archetypes (in dark blue) and their distorted copies (in bright blue). Comparing the values of these probabilities with the ones obtained in fig.8 for the ARCH RBM, we can see that these ones are higher even when using the same parameters and the same dataset. Since the topology of each model is different (i.e., the ARCH RBM has one hidden neuron per archetype, while the SRBM has a number of hidden neurons that can be adjusted), we cannot affirm that one model is strictly better than the other. However, in this particular case, the SRBM yields better classification probabilities.

Not only that, but each figure shows different trends. In the case of the ARCH RBM, as we provide the network with more training examples M, the accuracy in classifying them $\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma} = \boldsymbol{\eta}^{\mu,a})$ decreases, as if the ARCH RBM were prioritizing learning the underlying archetype over the given examples. This result is important because it challenges the idea behind learning discussed in Chapter 1. Which is that neural networks learn by adjusting parameters in order to minimize the error between the model and hidden probabilities. By doing so, they learn to make accurate

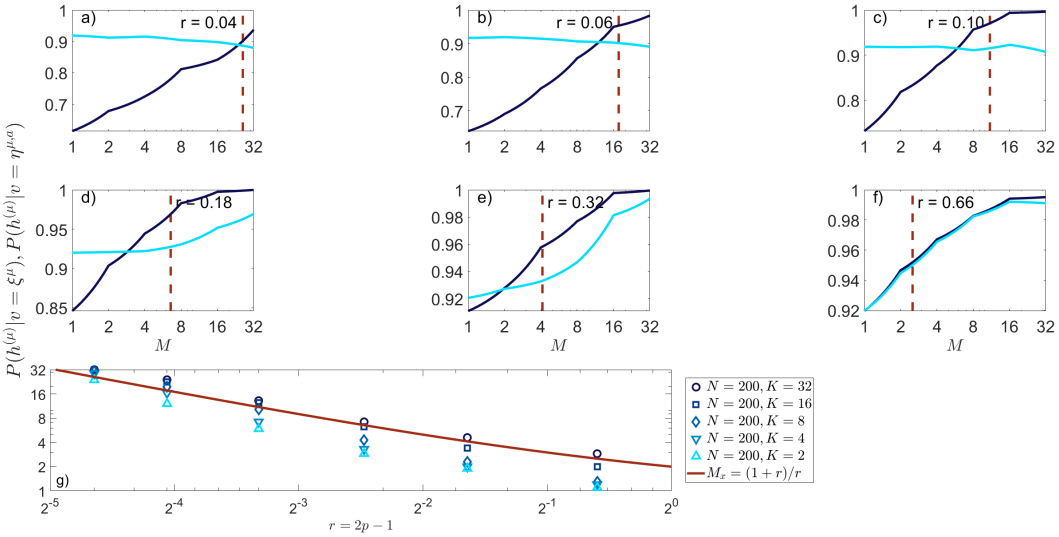


Figure 11: Numerical estimation of the crossover size M_x and its relation to α (average of 50 realizations). Other fixed parameters are $N = 200, N_h = 50, \epsilon = 10^{-3}, \lambda = 10^{-5}$.

predictions on new data.

In contrast, the data shown in the SRBM graph follows the opposite trend, particularly when $r > 0.18$, which is that increasing M leads to higher probabilities of classifying correctly both archetypes and examples. One possible explanation as to why the SRBM shows different behaviours depending on r is that, if r is not high enough, the examples from a certain archetype are more correlated to another class than to their own.

To see that, we can compute the *Simple Matching Coefficient* (SMC) [28], which is 1− *Hamming Distance* between two binary vectors (i.e., the number of equal bits). We calculate this value for each pair of examples in the training set and divide it by the number of visible units N.

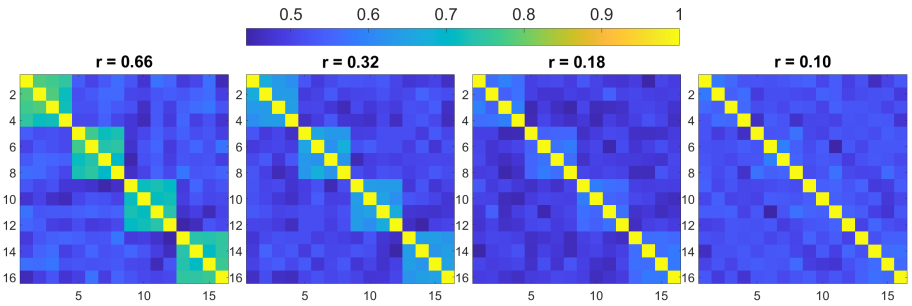


Figure 12: Similarity between training set examples

In Figure 12 we can observe this similarity coefficient between training set examples for $r \in \{0.66, 0.32, 0.18, 0.10, 0.06, 0.04\}$. We used $N = 200$ for the visible units and $K = 4$ different archetypes with $M = 4$ examples each. As we can see, we stop being able to distinguish the classes as soon as the quality of the dataset drops below $r = 0.32$, and in the case with $r \leq 0.10$, we cannot affirm that our examples can be divided into $K = 4$ distinct classes.

Returning to Figure 11 for $r \geq 0.18$, the fact that both classification probabilities increase with M also explains why the curves in fig.9 converge back to the horizontal line. Since once $\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma} = \boldsymbol{\xi}^\mu) = 1$, increasing further the number of examples per archetype makes $\mathcal{P}(\boldsymbol{z}^\mu|\boldsymbol{\sigma} = \boldsymbol{\eta}^{\mu,a})$ also tend to 1, thus the logarithm of their ratio tends to 0. In this case, the results are coherent with the goal of training in a neural network, since it manages to increase the classification probabilities on both the provided examples and the unseen archetypes.

Lastly, panel g in fig.11 reveals that the crossover size M_x seems to be correlated with the load of the SRBM α. This is illustrated by the data markers used for different values of α, which show the same tendency for all values of r (e.g., the bright upwards triangle has the lowest M_x value, while the dark blue circle has the highest).

Chapter 5

Conclusions

The main objective of this thesis has been to provide an in-depth study of supervised learning techniques for Restricted Boltzmann Machines (RBMs). In particular, we have investigated the alternative method presented by [1] and compared it with the Supervised Restricted Boltzmann Machine (SRBM). Throughout this study, we have also answered the questions posed in the outline of this thesis.

We started in Chapter 1 by introducing the most essential ideas behind RBMs, describing their architecture and defining the statistical principles that are used to implement these neural networks. In particular, we presented the Contrastive Divergence (CD) and Gibbs sampling algorithms that help approximate some calculations that are otherwise intractable.

Then, in Chapter 2, we addressed the main ideas regarding archetype-based learning, such as the concept of an archetype, as well as the key differences in the learning process between the standard implementation of an RBM and this archetype-based network. Using these concepts, we implemented an RBM model based in [1] that we denoted as ARCH RBM and used it for the rest of the thesis.

In Chapter 3 of the thesis, we analyzed the performance of the ARCH RBM model on several experiments and managed to obtain similar results to the ones shown in [1]. Specifically, our results indicate that there is a critical number of training examples for which the archetype RBM is effective at capturing the underlying patterns of a dataset. This threshold size is determined by the amount of distortion (that we denoted as r) applied to the archetypes in order to obtain the dataset

examples, and does not depend on the size of the network (i.e., the number of visible units N and the number of archetypes K).

Our main contributions are presented in Chapter 4, where we compared the performance of the proposed model with an SRBM by repeating the same experiments, and observed that the SRBM obtained better classification probabilities than the ARCH RBM in those particular experiments. With more research, we would be able to discern whether the difference in topology (i.e., the number of hidden neurons) affects these results.

Moreover, we obtained results that revealed that the two networks show a different behaviour in their learning. While the SRBM model manages to obtain high classification probabilities for the training data and the unseen archetypes, the ARCH RBM prioritizes making more accurate predictions on the archetypes, particularly when provided with sufficient examples.

Finally, our results indicate that the archetype-based model could be promising for classification and clustering problems. However, further research is needed to explore the model's performance on problems where the underlying archetypes may be correlated, such as the MNIST dataset of handwritten digits [26].

List of Figures

List of Tables

Bibliography

[1] Agliari, E., Alemanno, F., Barra, A. & De Marzo, G. The emergence of a concept in shallow neural networks. *Neural Networks* **148**, 232–253 (2022). URL https://www.sciencedirect.com/science/article/pii/S0893608022000272.

[2] Abiodun, O. I. *et al.* State-of-the-art in artificial neural network applications: A survey. *Heliyon* **4**, e00938 (2018). URL https://www.sciencedirect.com/science/article/pii/S2405844018332067.

[3] Kröse, B. & van der Smagt, P. *Fundamentals*, vol. 8, 11–20 (University of Amsterdam, 1996).

[4] Fawzi, A. *et al.* Discovering faster matrix multiplication algorithms with reinforcement learning. *Nature* **610**, 47–53 (2022).

[5] McCulloch, W. S. & Pitts, W. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics* **5**, 115–133 (1943). URL https://doi.org/10.1007/BF02478259.

[6] Smolensky, P. Information processing in dynamical systems: Foundations of harmony theory. Tech. Rep., Colorado Univ at Boulder Dept of Computer Science (1986).

[7] Hinton, G. E. Training products of experts by minimizing contrastive divergence. *Neural computation* **14**, 1771–1800 (2002).

[8] Hinton, G. E. & Salakhutdinov, R. R. Reducing the dimensionality of data with neural networks. *Science* **313**, 504–507 (2006).

[9] Salakhutdinov, R., Mnih, A. & Hinton, G. Restricted boltzmann machines for collaborative filtering. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, 791–798 (Association for Computing Machinery, New York, NY, USA, 2007). URL https://doi.org/10.1145/1273496.1273596.

[10] Larochelle, H. & Bengio, Y. Classification using discriminative restricted boltzmann machines. *Proceedings of the 25th international conference on Machine learning - ICML '08* (2008).

[11] Zheng, S. & Zhao, J. States identification of complex chemical process based on unsupervised learning. In Eden, M. R., Ierapetritou, M. G. & Towler, G. P. (eds.) *13th International Symposium on Process Systems Engineering (PSE 2018)*, vol. 44 of *Computer Aided Chemical Engineering*, 2239–2244 (Elsevier, 2018). URL https://www.sciencedirect.com/science/article/pii/B9780444642417503682.

[12] Tieleman, T. Training restricted boltzmann machines using approximations to the likelihood gradient. *Proceedings of the 25th international conference on Machine learning - ICML '08* (2008).

[13] Cai, X., Xu, Z., Lai, G., Wu, C. & Lin, X. Gpu-accelerated restricted boltzmann machine for collaborative filtering. In *GPU-Accelerated Restricted Boltzmann Machine for Collaborative Filtering*, vol. 7439, 303–316 (2012).

[14] Fischer, A. & Igel, C. Training restricted boltzmann machines: An introduction. *Pattern Recognition* **47**, 25–39 (2014). URL https://www.sciencedirect.com/science/article/pii/S0031320313002495.

[15] J., H. *et al. Introduction To The Theory Of Neural Computation*, vol. 44 (Westview Press, 1991).

[16] Huang, H. Statistical mechanics of restricted boltzmann machine. *Statistical Mechanics of Neural Networks* 111–132 (2021). URL https://doi.org/10.1007/978-981-16-7570-6_10.

[17] Upadhya, V. & Sastry, P. S. An overview of restricted boltzmann machines. *Journal of the Indian Institute of Science* **99**, 225–236 (2019). URL `https://doi.org/10.1007/s41745-019-0102-z`.

[18] Hinton, G. A practical guide to training restricted boltzmann machines (version 1). *Technical Report UTML TR 2010-003, University of Toronto* **9** (2010).

[19] Suvrit, S., Bottou, L. & Bousquet, O. *The Tradeoffs of Large-Scale Learning* (PHI LEARNING, 2015).

[20] Bilmes, J., Asanovic, K., Chin, C.-W. & Demmel, J. Using phipac to speed error back-propagation learning. *1997 IEEE International Conference on Acoustics, Speech, and Signal Processing* **5**, 4153–4156 vol.5 (1997).

[21] Meyn, S. P. & Tweedie, R. L. *Markov models* (Springer, 1994).

[22] Pierre, B. *Recurrence and Ergodicity. Positive recurrence.*, 104–105 (Springer, 2001).

[23] Pierre, B. *Gibbs Sampler. Convergence Rate of the Gibbs Sampler.*, 288–289 (Springer, 2001).

[24] Bengio, Y. & Delalleau, O. Justifying and generalizing contrastive divergence. *Neural Computation* **21**, 1601–1621 (2009).

[25] Larochelle, H. & Bengio, Y. Classification using discriminative restricted boltzmann machines. *Proceedings of the 25th International Conference on Machine Learning* 536–543 (2008).

[26] Deng, L. The mnist database of handwritten digit images for machine learning research. *IEEE Signal Processing Magazine* **29**, 141–142 (2012).

[27] Bowers, J. S. What is a grandmother cell? and how would you know if you found one? *Connection Science* **23**, 91–95 (2011). URL `https://doi.org/10.1080/09540091.2011.568608`. `https://doi.org/10.1080/09540091.2011.568608`.

[28] Daughtery, S. Measures of similarity in data (2010). URL `http://mines.humanoriented.com/classes/2010/fall/csci568/portfolio_exports/sdaugherty/similarity.html`.