UNIVERSITÀ DEGLI STUDI DI PADOVA

Dipartimento di Fisica e Astronomia "Galileo Galilei"

Master Degree in Physics of Data

Final Dissertation

Study of performances for

Restricted Boltzmann Machines

Thesis supervisor

Prof. Marco Baiesi

Candidate

Matteo Bortoletto

Academic Year 2020/2021

To my parents Patrizia and Massimo and to my sister Elisa.

Abstract

Restricted Boltzmann Machines (RBMs) are one of the most relevant unsupervised learning methods. The aim of this thesis is to study their performances as a function of their parameters. First, we consider binary-valued RBMs and then we introduce the so-called centering trick, which is known to solve the absence of invariance to flip transformations. Moreover, centering also leads to more accurate models. Then, we discuss RBMs with real-valued units. In particular, we focus on rectified linear units, which are able to achieve better generative performances than binary units.

Contents

1 Introduction **1**

2 Probabilistic graphical models **3**

2.1 Conditional independence and factorization 3

2.2 Markov random fields 4

2.3 Unsupervised Markov random field learning 6

2.3.1 Gradient ascent 8

2.3.2 Log-likelihood gradient of MRFs with hidden units 8

2.4 Markov chains and Gibbs sampling 9

2.4.1 Markov chains 10

2.4.2 Gibbs sampling 11

3 Restricted Boltzmann Machines **13**

3.1 Binary-binary Restricted Boltzmann Machines 13

3.2 Correlations 14

3.3 Conditional Distributions 15

3.4 Block Gibbs sampling 17

3.5 The gradient of the log-likelihood 17

3.6 Training Restricted Boltzmann Machines 19

3.6.1 Contrastive Divergence 19

3.6.2 Persistent Contrastive Divergence 22

3.6.3 Parallel tempering 23

3.7 Practical Considerations 24

4 Centered Restricted Boltzmann Machines **27**

4.1 The centering trick 28

4.2 Training centered Restricted Boltzmann Machines 29

4.2.1 Centering the gradient 30

5 Restricted Boltzmann Machines with real-valued units **33**
5.1 Gaussian units . . . 33
5.2 Binomial units . . . 34
5.3 Rectified linear units . . . 35
5.3.1 Emergence of compositional representations . . . 36

6 Experiments **37**
6.1 Setup . . . 37
6.1.1 Datasets and setup . . . 37
6.2 Normal binary-binary RBMs . . . 38
6.2.1 Comparison between CD, PCD and PT . . . 38
6.2.2 Receptive fields . . . 39
6.2.3 Generative performances . . . 40
6.3 Centered binary-binary RBMs . . . 41
6.4 Binary-ReLU RBMs . . . 43

7 Conclusions **47**

Bibliography **49**

1 Introduction

Deep learning, a family of machine learning algorithms based on artificial neural networks, has dramatically improved state-of-the-art performances in numerous fields, including image processing, natural language processing, speech recognition, intelligent gaming, automated transportation, healthcare and genomics. These achievements are obtained thanks to the large amount of data and computational power that is available nowadays. However, the theoretical understanding of such deep models evolves much slower than their increasing complexity, so these networks behave more and more as black-boxes. Using simpler models allows us to give a better interpretation of their outputs. In this thesis we consider Restricted Boltzmann Machines, which are much simpler than deep models but still can learn very efficient representations of complex data.

Restricted Boltzmann Machines (RBMs) are artificial neural networks that represent some of the most common building blocks of deep probabilistic models. These models can learn, in some way, probability distributions over multiple variables. In particular, RBMs are a particular kind of Boltzmann Machines (BM), which are energy-based stochastic recurrent neural networks. With respect to a BM, an RBM is not recurrent, and this makes its learning procedure particularly simple and efficient.

In this work we present the theoretical framework to which RBMs belong and we study the performances of various types of models. Chapter 2 provides an overview of probabilistic graphical models. In particular, we focus on Markov random fields, of which RBMs are a particular kind. Then, we discuss how such models are able to learn a probability distribution in an unsupervised way and how to draw samples from it. In Chapter 3 we introduce RBMs with binary units and we analyse their properties, which mainly follow from the absence of intra-layer connections. Finally, we discuss how to train RBMs by presenting the main techniques: contrastive divergence, persistent contrastive divergence and parallel tempering. In Chapter 4, following the work of Melchior et al. [26], we discuss the so-called centering trick, which makes RBMs flip-invariant and also improves their performances. Next, in Chapter 5, starting from the work of Nair and Hinton [28], we introduce RBMs with real-valued units. In particular, we focus on rectified linear units, which are known for improving learning and generative performances.

Chapter 6 presents some experiments that study RBMs performances. First we consider normal binary-binary RBMs, then we move on to centered binary-binary RBM and lastly we analyse binary-ReLU RBMs.

2 Probabilistic graphical models

Probabilistic Graphical Models (PGMs) are a rich framework that use a graph-based representation as the basis for compactly encoding a complex distribution over a high-dimensional space [19]. In this graphical representation the nodes correspond to the variables in our domain, and the edges correspond to probabilistic interactions between them. An example is shown in Figure 2.1.

PGMs offer several useful properties. First, they provide a simple way to visualize the structure of probabilistic models and can be used to design and motivate new ones. Then, by inspecting the graph, we can get information about properties of a model. Lastly, complex computations can be expressed in terms of graphical manipulations which ease the mathematical treatment.

2.1 Conditional independence and factorization

An important property for probability distributions over multiple variables is that of conditional independence [3]. Consider three variables $\boldsymbol{x}$, $\boldsymbol{y}$ and $\boldsymbol{z}$ and suppose that the conditional probability distribution of $\boldsymbol{x}$ given $\boldsymbol{y}$ and $\boldsymbol{z}$ is independent of $\boldsymbol{y}$. Then, we can write

$$p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z}), \tag{2.1}$$

and we say that $\boldsymbol{x}$ is conditionally independent of $\boldsymbol{y}$ given $\boldsymbol{z}$. This can also be expressed in the form

$$p(\boldsymbol{x}, \boldsymbol{y}|\boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{y}, \boldsymbol{z})p(\boldsymbol{y}|\boldsymbol{z}) = p(\boldsymbol{x}|\boldsymbol{z})p(\boldsymbol{y}|\boldsymbol{z}). \tag{2.2}$$

This means that the joint distribution of $\boldsymbol{x}$ and $\boldsymbol{y}$ (conditioned on $\boldsymbol{z}$) factorizes into the product of the marginal distribution of $\boldsymbol{x}$ and the marginal distribution of $\boldsymbol{y}$ (conditioned on $\boldsymbol{z}$). Thus, given $\boldsymbol{z}$, $\boldsymbol{x}$ and $\boldsymbol{y}$ are statistically independent.

An elegant feature of graphical models is that conditional independence properties of the joint distribution can be read directly from the graph without having to perform any analytical

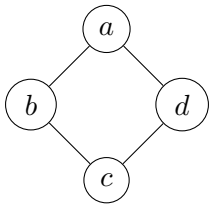


Figure 2.1: Example of probabilistic graphical model.

manipulations [3]. In fact, suppose to have an undirected graph with three sets of nodes A, B and C. If we want to verify if A is independent of B we just have to consider all possible paths that connect nodes in A to nodes in B. If all such paths pass through one or more nodes in C, then the conditional independence property holds. Another important feature is that the graph defines a skeleton for compactly representing a high-dimensional distribution: we can "break up" the distribution into independent factors and then define the overall joint distribution as a product of these factors [19]. For example, the factorization of the distribution associated with the graph in Figure 2.1 is $p(a, b, c, d) = \frac{1}{Z}\psi_1(a, b)\psi_2(b, c)\psi_3(c, d)\psi_4(a, d)$. These two features are deeply connected. Indeed, the independence properties of the distribution are precisely what allow it to be represented in a factorized form. Conversely, a particular factorization of the distribution guarantees that certain independencies hold. These properties allow complex computations (e.g., marginalization) to be derived efficiently by using algorithms that exploit the graph structure.

Different types of graphical models are associated with different kinds of graph structures. For example, *Bayesian networks* are associated with directed graphs, i.e. graphs in which the edges have a direction. *Markov random fields*, also known as Markov networks, are associated with undirected graphs, where edges are bidirectional. Restricted Boltzmann Machines are a particular kind of Markov random field, hence we will focus on this graph structure. In the following, we adopt the notation used by Fischer and Igel in their article on training Restricted Boltzmann Machines [12].

2.2 Markov random fields

An RBM is a particular kind of Markov random field, which is an undirected graph. An *undirected graph* is an ordered pair $G = (V, E)$, where V is a finite set of nodes and E is a set of undirected edges [13]. An edge consists of a pair of nodes from V and the *neighborhood* $\mathcal{N}_v = \{w \in V : \{w, v\} \in E\}$ of a node v is defined by the set of nodes connected to v. An example of undirected graph is shown in Figure 2.2.

A *clique* is a subset of V such that there exists a link between all pairs of nodes in the subset [3]. A clique is called *maximal* if no node can be added such that the resulting set is still a clique. In the undirected graph in Figure 2.2, both $\{v_1, v_2\}$ and $\{v_1, v_2, v_3\}$ are cliques but only the latter is maximal. We denote by $\mathcal{C}$ the set of all maximal cliques of an undirected graph. In this section, we show that the factors in the decomposition of the joint distribution over the

graph can be defined as functions of the variables in the cliques.

A sequence of nodes $v_1, v_2, \ldots, v_m \in V$, with $\{v_i, v_{i+1}\} \in E$ for $i = 1, \ldots, m-1$ defines a *path* from v_1 to v_m and we say that a set $\mathcal{V} \subset V$ *separates* two nodes $v \notin \mathcal{V}$ and $w \notin \mathcal{V}$ if every path from v to w contains a node from $\mathcal{V}$. For example, in Figure 2.2 $\mathcal{V} = \{v_4, v_5\}$ separates v_1 and v_8.

Given an undirected graph $G = (V, E)$, we associate each node $v \in V$ to a random variable X_v taking values in a state space Λ_v. For simplicity, we assume $\Lambda_v = \Lambda$ for all $v \in V$. The set of random variables $X = (X_v)_{v \in V}$ is called a *Markov random field* (MRF) if the joint probability distribution p fulfills the *local Markov property* with respect to the graph [13].

Definition 1 (Local Markov property)**.** *The joint probability distribution of a set of random variables $X = (X_v)_{v \in V}$ is said to fulfill the local Markov property with respect to a graph $G = (V, E)$ if for all $v \in V$ the random variable X_v is conditionally independent of all other variables given its neighborhood $(X_w)_{w \in \mathcal{N}_v}$. That is, for all $v \in V$ and all $x \in \Lambda^{|V|}$, one has that*

$$p(x_v | (x_w)_{w \in V \setminus \{v\}}) = p(x_v | (x_w)_{w \in \mathcal{N}_v}). \tag{2.3}$$

If the probability distribution of the MRF is strictly positive, the local Markov property is equivalent to other two types of Markov property [13].

Definition 2 (Global Markov property)**.** *The MRF is said to have the* global Markov property *with respect to a graph $G = (V, E)$ if for any three disjunct subsets $A, B, S \subset V$, such that all nodes in A and B are separated by S, the variables $(X_a)_{a \in A}$ and $(X_b)_{b \in B}$ are conditionally independent given $(X_s)_{s \in S}$, i.e., for all $x \in \Lambda^{|V|}$ one has that*

$$p((x_a)_{a \in A | (x_t)_{t \in S \cup B}}) = p((x_a)_{a \in A | (x_t)_{t \in S}}). \tag{2.4}$$

Definition 3 (Pairwise Markov property)**.** *The MRF is said to have the* pairwise Markov property *with respect to a graph $G = (V, E)$ if any two non-adjacent variables are conditionally independent given all other variables: if $\{v, w\} \notin E$, then $p(x_v, x_w | (x_t)_{t \in V \setminus \{v,w\}}) = p(x_v | (x_t)_{t \in V \setminus \{v,w\}}) p(x_w | (x_t)_{t \in V \setminus \{v,w\}})$ for all $x \in \Lambda^{|V|}$.*

We know that conditional independence of random variables and factorization properties of the joint probability distribution are closely related, but we still have to formally express their connection. The following theorem shows that if the Markov property is satisfied there exists a general factorization of MRF distributions [7].

Theorem 1 (Hammersley-Clifford)**.** *A strictly positive distribution p satisfies the Markov property with respect to an undirected graph G if and only if p factorizes over G.*

A distribution is said to *factorize* over an undirected graph G with maximal cliques $\mathcal{C}$ if there

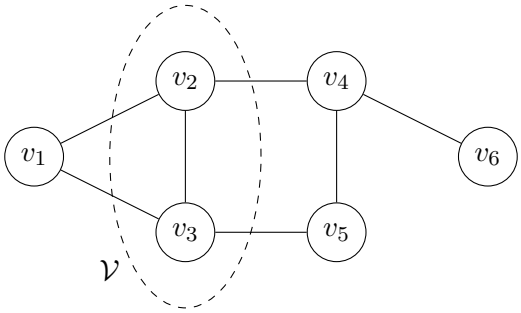


Figure 2.2: Example of undirected graph. Here the neighborhood of node v_4 is $\{v_2, v_5, v_6\}$. Both $\{v_1, v_2\}$ and $\{v_1, v_2, v_3\}$ are cliques but only the latter is maximal. $\mathcal{V} = \{v_2, v_3\}$ separates v_1 and v_6.

exists a set of non-negative functions $\{\psi \in C\}_{C \subset \mathcal{C}}$, called *potential functions*, satisfying

$$\forall \boldsymbol{x}_C, \hat{\boldsymbol{x}}_C \in \Lambda^{|V|} : (x_c)_{c \in C} = (\hat{x}_c)_{c \in C} \quad \Rightarrow \quad \psi_C(\boldsymbol{x}_C) = \psi_C(\hat{\boldsymbol{x}}_C), \tag{2.5}$$

and

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C), \tag{2.6}$$

where the normalization constant Z is called *partition function* [3]. If p is strictly positive, also the potential functions are strictly positive. Thus, we can write

$$p(\boldsymbol{x}) = \frac{1}{Z} \prod_{C \in \mathcal{C}} \psi_C(\boldsymbol{x}_C) = \frac{1}{Z} \exp\left(\sum_{C \in \mathcal{C}} \ln \psi_C(\boldsymbol{x}_C) \right) = \frac{e^{-E(\boldsymbol{x}_C)}}{Z}, \tag{2.7}$$

where $E = \sum_{C \in \mathcal{C}} \ln \psi_C(\boldsymbol{x}_C)$ is called the *energy function* [3]. Therefore, the (strictly positive) joint probability distribution of every MRF can be written as product of potentials using Equation (2.7) – which is referred to as the *Gibbs distribution* – for which the total energy is obtained by adding the energy of each of the maximal cliques.

In an undirected graph, the potentials do not have a specific probabilistic interpretation [3]. This is in contrast to directed graphs in which each factor represents the conditional distribution of the corresponding variable. One consequence is that their product will in general not be correctly normalized. This gives flexibility in choosing the potential functions, but it raises the question of how to motivate a choice of potential function for a particular application. A possible interpretation is to see the potential function as expressing which configurations of the local variables are preferred to others [3].

2.3 Unsupervised Markov random field learning

In *unsupervised learning* we want to learn an unknown distribution q from unlabeled data. The hope is that, through mimicry, the machine is forced to build a compact internal representation of its world and then generate imaginative content. If we assume that the structure of the

graphical model is known and that the energy function is parameterized by $\boldsymbol{\theta}$, unsupervised learning of a data distribution with an MRF means adjusting the parameters $\boldsymbol{\theta}$. Therefore, we adapt the notation $p(x|\boldsymbol{\theta})$ to emphasize the dependency of a distribution on its parameters.

Let us consider a training set $S = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_l\}$, in which the samples are unlabeled and assumed to be independent and identically distributed with unknown distribution q. The common way to learn the model parameters is *maximum–likelihood estimation* [13], which in our case corresponds to finding the MRF parameters $\boldsymbol{\theta}$ that maximize the likelihood given the training data, defined as

$$\mathcal{L}(\boldsymbol{\theta}|S) = \prod_{i=1}^{l} p(\boldsymbol{x}_i|\boldsymbol{\theta}). \tag{2.8}$$

Since the logarithm is a monotonically increasing function, maximizing the likelihood is the same as maximizing the log-likelihood, which is given by

$$\ln \mathcal{L}(\boldsymbol{\theta}|S) = \ln \prod_{i=1}^{l} p(\boldsymbol{x}_i|\boldsymbol{\theta}) = \sum_{i=1}^{l} \ln p(\boldsymbol{x}_i|\boldsymbol{\theta}). \tag{2.9}$$

From a computational standpoint, maximizing the log-likelihood is less expensive and avoids numerical errors. However, in general it is not possible to find the maximum likelihood parameters analytically for the Gibbs distribution of a MRF, since it requires to compute the partition function Z. Thus, we need to use numerical approximations, for example gradient ascent.

Maximizing the likelihood corresponds to minimizing the distance between the unknown distribution q underlying S and the distribution p of the MRF in terms of the *Kullback–Leibler divergence* (KL divergence) [20], which for a finite state space Ω is given by

$$\mathrm{KL}(q||p) = \sum_{\boldsymbol{x}\in\Omega} q(\boldsymbol{x}) \ln \frac{q(\boldsymbol{x})}{p(\boldsymbol{x})} = \sum_{\boldsymbol{x}\in\Omega} q(\boldsymbol{x}) \ln q(\boldsymbol{x}) - \sum_{\boldsymbol{x}\in\Omega} q(\boldsymbol{x}) \ln p(\boldsymbol{x}). \tag{2.10}$$

The KL divergence is a (non-symmetric) measure of the difference between two distributions. A KL divergence of zero indicates that the two distributions in question have identical quantities of information, i.e. they are the same. In the other cases it is always positive. From Equation (2.10) we see that the KL divergence can be expressed as the difference between the Shannon entropy of q and a second term. Only the latter depends on the parameters subject to optimization, which is the log-likelihood. Therefore, maximizing the log-likelihood corresponds to minimizing the KL divergence.

Recall that $p(\boldsymbol{x}) \propto e^{-E(\boldsymbol{x})}$, therefore the probability of a configuration is inversely proportional to its energy. Therefore, from a physical perspective, learning can be interpreted as tuning the model parameters such that the energy corresponding to our input patterns is minimized.

2.3.1 Gradient ascent

As mentioned above, in general it is not possible to find parameters by maximizing the log-likelihood analytically. Thus, numerical approximations are needed. The standard technique is *gradient ascent* on the log-likelihood [13, 16, 25]. This corresponds to iteratively updating the parameters $\boldsymbol{\theta}^{(t)}$ to $\boldsymbol{\theta}^{(t+1)}$ based on the gradient of the log-likelihood. The complete form of the update rule is the following:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \eta \frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}^{(t)}|S)}{\partial \boldsymbol{\theta}^{(t)}} - \lambda \boldsymbol{\theta}^{(t)} + \nu \Delta \boldsymbol{\theta}^{(t-1)}. \tag{2.11}$$

The parameter $\eta \in \mathbb{R}^+$ is the learning rate, λ is the weight decay parameter, ν is the momentum parameter and $\Delta\boldsymbol{\theta}^{(t-1)}$ is the gradient at step $t-1$. If $\lambda \in \mathbb{R}_0^+$ and $\nu \in \mathbb{R}_0^+$ are set to zero, we have the vanilla gradient ascent. In general, it is better to have models with small weights in absolute value. To achieve this, we can consider an objective function in which we subtract to the log-likelihood half of the norm of the parameters $1/2\|\theta\|^2$, weighted by λ. This method is called *weight decay*, it penalizes weights with large magnitude and it leads to the $-\lambda\boldsymbol{\theta}^{(t)}$ term in the update rule (2.11). The update rule can be further extended by a *momentum* term [30], $\Delta\boldsymbol{\theta}^{(t-1)}$, weighted by the parameter ν, which serves as a memory of the direction we are moving in the parameter space. Using momentum helps against oscillations in the iterative update procedure and may speed up the learning process.

2.3.2 Log-likelihood gradient of MRFs with hidden units

Let us assume that our goal is to model a m-dimensional unknown probability distribution q. Usually, $\boldsymbol{X}$ is split into *visible* (or observed) variables $V = (V_1, \ldots, V_m)$ corresponding to the components of the observations and *hidden* (or latent) variables $H = (H_1, \ldots, H_n)$ given by the remaining $n = |X| - m$ variables. While the visible variables correspond to the components of an observation, the hidden variables introduce dependencies between the visible variables. For example, if our data consist of images, the visible variables correspond to the pixels intensity whereas the hidden variables introduce correlations between them. Thus, using latent variables we can describe complex distributions over the visible variables by means of conditional distributions. The joint probability distribution of (V, E) is described by the Gibbs distribution. Usually we are interested in the probability distribution of V, which can be obtained by marginalizing over $\boldsymbol{h}$:

$$p(\boldsymbol{v}) = \sum_{\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}, \tag{2.12}$$

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

MRFs learning algorithms are based on gradient ascent on the log-likelihood. For a model of

the form (2.12) with parameters $\boldsymbol{\theta}$, the log-likelihood given a single training example $\boldsymbol{v}$ is

$$\begin{aligned}\ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v}) = \ln p(\boldsymbol{v}|\boldsymbol{\theta}) &= \ln \frac{1}{Z} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \\ &= \ln \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} - \ln Z \\ &= \ln \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} - \ln \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}, \end{aligned} \tag{2.13}$$

and the gradient is

$$\begin{aligned}\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial \boldsymbol{\theta}} &= \frac{\partial}{\partial \boldsymbol{\theta}} \left(\ln \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \right) - \frac{\partial}{\partial \boldsymbol{\theta}} \left(\ln \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \right) \\ &= -\frac{1}{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} + \frac{1}{\sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} \\ &= -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} + \sum_{\boldsymbol{v},\boldsymbol{h}} p(\boldsymbol{v},\boldsymbol{h}) \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}}, \end{aligned} \tag{2.14}$$

where in the last step we used the fact that

$$p(\boldsymbol{h}|\boldsymbol{v}) = \frac{p(\boldsymbol{v},\boldsymbol{h})}{p(\boldsymbol{v})} = \frac{\frac{1}{Z} e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\frac{1}{Z} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}} = \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{\sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})}}. \tag{2.15}$$

The last expression of (2.14) is the difference between two expectations: the first is the expected value of the energy function under the model distribution and the second is the expected value under the conditional distribution of the hidden variables given the training example. This is a well-known decomposition into the *positive phase* and *negative phase* of learning [16]. Intuitively, the positive phase can be interpreted as pushing down on the energy of training examples and the negative phase as pushing up on the energy of samples drawn from the model[1].

The drawback of such expression is that, in general, the computation of these sums is exponential in the number of variables of the MRF and thus for huge models it is not feasible [13, 16]. Therefore, we need to use approximations. The standard procedure to compute these expectations is by using samples drawn from the corresponding distribution. Such techniques are called Markov chain Monte Carlo algorithms.

2.4 Markov chains and Gibbs sampling

Markov chains have a key role in MRF training because they provide a method to draw samples from non-trivial probability distributions such as the Gibbs distribution. In particular,

[1] Essentially, the negative phase acts to reduce the probability of the samples drawn from the model distribution. In this sense, these samples can be considered to represent the model "incorrect beliefs about the world" and they are often referred to as *fantasy particles*. Actually, the negative phase has been proposed as a possible explanation for dream sleep [8]. However, neuroscientific experiments do not seem to support this hypothesis.

Restricted Boltzmann Machines – the focus of this thesis – are trained by using Gibbs sampling, which will be discussed in this section.

2.4.1 Markov chains

A *Markov chain* is a stochastic process describing a sequence of possible events in which the probability of each event depends only on the state attained in the previous event [4, 19]. Formally, a Markov chain is a family of random variables $X = \{X^{(k)} | k \in \mathbb{N}_0\}$ taking values in a (finite) set Ω and for which $\forall k \geq 0$ and $\forall j, i, i_0, \ldots, i_{k-1} \in \Omega$ one has

$$\begin{aligned} p_{ij}^{(k)} &= \Pr\left(X^{(k+1)} = j | X^{(k)} = i, X^{(k-1)} = i_{k-1}, \ldots, X^{(0)} = i_0\right) \\ &= \Pr\left(X^{(k+1)} = j | X^{(k)} = i\right). \end{aligned} \tag{2.16}$$

This is also referred to as Markov property, but it considers temporal neighborhood, while the Markov properties discussed in Section 2.2 consider neighborhood induced by the graph topology. If for all points in time $k \geq 0$ the $p_{ij}^{(k)}$ have the same value p_{ij} – i.e. the transition probabilities do not change over time – the chain is called *homogeneous* and the matrix $\mathbf{P} = (p_{ij})_{i,j \in \Omega}$ is called the *transition matrix* of the homogeneous Markov chain [3].

If the initial distribution $\mu^{(0)}$ (the probability distribution of $X^{(0)}$) is given by the probability vector $\boldsymbol{\mu}^{(0)} = (\mu^{(0)}(i))_{i \in \Omega}$, with $\mu^{(0)}(i) = \Pr(X^{(0)} = i)$, the distribution $\mu^{(k)}$ of $X^{(k)}$ is given by $\boldsymbol{\mu}^{(k)T} = \boldsymbol{\mu}^{(0)T}\mathbf{P}^k$. A distribution π for which $\boldsymbol{\pi}^T = \boldsymbol{\pi}^T\mathbf{P}$ is called a *stationary distribution* or *equilibrium distribution*. If the Markov chain at time k has reached the stationary distribution $\boldsymbol{\mu}^{(k)} = \boldsymbol{\pi}$, then all subsequent states will have the same distribution $\boldsymbol{\pi}$, that is $\boldsymbol{\mu}^{(k+n)} = \boldsymbol{\pi}$ for all $n \in N$. A sufficient but not necessary condition for a distribution π to be stationary with respect to a Markov chain described by the transition probabilities p_{ij}, $i, j \in \Omega$, is that $\forall i, j \in \Omega$

$$\pi(i)p_{ij} = \pi(j)p_{ji}. \tag{2.17}$$

This is called *detailed balance condition* and it implies that, around any closed cycle of states, there is no net flow of probability. A Markov chain that satisfies this property is said to be *reversible* [3].

Of particular interest are Markov chains for which there exists a unique stationary distribution. For a finite state space Ω, this happens if the Markov chain is irreducible. A Markov chain is *irreducible* or *ergodic* if one can go from any state in Ω to any other in a finite number of transitions or, more formally, if $\forall i, j \in \Omega\ \exists k > 0$ with $\Pr(X^{(k)} = j | X^{(0)} = i) > 0$.

Finally, a chain is called *aperiodic* if every state can reoccur at irregular times. Formally, a chain is aperiodic if for all $i \in \Omega$ the greatest common divisor of all elements in the set $\{k \in N_0 | \Pr(X^{(k)} = i | X^{(0)} = i) > 0\}$ is 1. It is possible to prove that an irreducible and aperiodic Markov chain on a finite state space is guaranteed to converge to its stationary distribution. The theorem requires the notion of distance of variation between two probability distributions. If α and β are two distributions defined on a finite state space Ω, the *distance of variation* is

defined as [19]

$$d_V(\boldsymbol{\alpha}, \boldsymbol{\beta}) = \frac{1}{2}|\boldsymbol{\alpha} - \boldsymbol{\beta}| = \frac{1}{2}\sum_{x\in\Omega}|\alpha(x) - \beta(x)|. \tag{2.18}$$

Now, we can enunciate the following theorem [4].

Theorem 2. *Let π be the stationary distribution of an irreducible and aperiodic Markov chain on a finite state space with transition matrix $\mathbf{P}$. For an arbitrary starting distribution μ*

$$\lim_{k\to\infty} d_V(\boldsymbol{\mu}^T\mathbf{P}^k, \boldsymbol{\pi}^T) = 0. \tag{2.19}$$

Markov chain Monte Carlo methods (MCMC) make use of this convergence theorem for producing samples. Suppose we want to sample from a distribution q with a finite state space. We just have to construct an irreducible and aperiodic Markov chain with stationary distribution $\pi = q$ and if k is large enough, the state $x^{(k)}$ of $X^{(k)}$ from the constructed chain can be approximately considered as a sample from q. Gibbs sampling is such a MCMC method.

2.4.2 Gibbs sampling

Gibbs sampling [14] belongs to the broader class of Metropolis-Hastings algorithms, which are MCMC algorithms that generate the transitions of a Markov chain in two substeps. In the first substep, a candidate state is picked at random from a so-called proposal distribution. In the second substep, the candidate state is accepted as the new state of the Markov chain with an acceptance probability ensuring that detailed balance holds. In particular, Gibbs sampling constructs a Markov chain by updating each variable based on its conditional distribution given the state of all the other variables. This is the standard technique that is used to produce approximate samples from the Gibbs distribution of an MRF.

Let us consider a MRF $\boldsymbol{X} = (X_1, \ldots, X_N)$ represented by an undirected graph $G = (V, E)$, where to simplify the notation we set $V = \{1, \ldots, N\}$. Assuming that the MRF changes its state over time, we obtain a Markov chain $X = \{\boldsymbol{X}^{(k)} | k \in \mathbb{N}_0\}$, which takes values in $\Omega = \Lambda^N$. A new state of the chain is produced as follows [13]:

1. first, we pick at random X_i, $i \in V$ with probability $q(i)$ given by a probability distribution q on V;
2. then, the new state for X_i is sampled based on its conditional probability distribution given the state $(x_v)_{v\in V\setminus i}$ of all other variables $(X_v)_{v\in V\setminus i}$, which is $\pi(\boldsymbol{x}_i|(x_v)_{v\in V\setminus i}) = \pi(x_i|(x_w)_{w\in\mathcal{N}_i})$ because of the local Markov property of MRFs.

The transition probability $p_{\boldsymbol{xy}}$ for two states $\boldsymbol{x}, \boldsymbol{y}$ of the MRF $\boldsymbol{X}$ with $\boldsymbol{x} \neq \boldsymbol{y}$ is

$$p_{\boldsymbol{xy}} = \begin{cases} q(i)\pi(y_i|(x_v)_{v\in V\setminus i}) & \text{if } \exists i \in V \text{ so that } \forall v \in V \text{ with } v \neq i : x_v = y_v \\ 0 & \text{otherwise} \end{cases}, \tag{2.20}$$

and in particular the probability that the state of the MRF stays the same is

$$p_{\boldsymbol{xx}} = \sum_{i \in V} q(i)\pi(x_i|(x_v)_{v \in V \setminus i}). \tag{2.21}$$

Following Theorem 2, in order to show that the Markov chain defined by these transition probabilities converges to the joint distribution π of the MRF, we have to prove that π is the stationary distribution of the Gibbs chain and that the chain is irreducible and aperiodic. To prove that π is the stationary distribution we just have to verify that the detailed balance condition (2.17) holds. For $\boldsymbol{x} = \boldsymbol{y}$ this follows directly. If $\boldsymbol{x}$ and $\boldsymbol{y}$ differ in the value of more than one random variable, then this follows from the fact that, according to Equation (2.20), $p_{\boldsymbol{yx}} = p_{\boldsymbol{xy}} = 0$. Finally, if $\boldsymbol{x}$ and $\boldsymbol{y}$ differ only in the state of exactly one variable X_i, i.e., $y_j = x_j$ for $j \neq i$ and $y_i \neq x_i$, we have:

$$\begin{aligned}
\pi(\boldsymbol{x})p_{\boldsymbol{xy}} &= \pi(\boldsymbol{x})q(i)\pi(y_i|(x_v)_{v \in V \setminus i}) \\
&= \pi(x_i|(x_v)_{v \in V \setminus i})q(i)\frac{\pi(y_i, (x_v)_{v \in V \setminus i})}{\pi((x_v)_{v \in V \setminus i})} \\
&= \pi(y_i|(x_v)_{v \in V \setminus i})q(i)\frac{\pi(x_i, (x_v)_{v \in V \setminus i})}{\pi((x_v)_{v \in V \setminus i})} \\
&= \pi(\boldsymbol{y})q(i)\pi(x_i, (x_v)_{v \in V \setminus i}) \\
&= \pi(\boldsymbol{y})p_{\boldsymbol{yx}}.
\end{aligned} \tag{2.22}$$

Thus, the detailed balance condition is fulfilled and π is the stationary distribution.

Since π is strictly positive, so are the conditional probability distributions of the single variables. This means that every single variable X_i can take every state $x_i \in \Lambda$ in a single transition step and thus every state of the whole MRF can reach any other in Λ^N in a finite number of steps. Thus, the Markov chain is irreducible. Moreover, since $p_{\boldsymbol{xx}} > 0$ for all $\boldsymbol{x} \in \Lambda^N$, the Markov chain is aperiodic. Therefore, Theorem 2 assures us that the chain converges to the stationary distribution π.

3 Restricted Boltzmann Machines

3.1 Binary-binary Restricted Boltzmann Machines

Invented under the name *harmonium* by Paul Smolensky in 1986 [33], Restricted Boltzmann Machines (RBMs) are MRFs containing a layer of m observable variables $\boldsymbol{V} = (V_1, \ldots, V_m)$ and a single layer of n hidden variables $\boldsymbol{H} = (H_1, \ldots, H_n)$, which represent the dependencies between the visible units. The key property of such models is that they have no intra-layer connections. Figure 3.1 shows the graph structure of an RBM.

In binary-binary RBMs, $(\boldsymbol{V}, \boldsymbol{H})$ take values $(\boldsymbol{v}, \boldsymbol{h}) \in \{0,1\}^{m+n}$ and the joint probability distribution under the model is given by the Gibbs distribution

$$p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} \exp(-E(\boldsymbol{v}, \boldsymbol{h})), \tag{3.1}$$

with energy function

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\boldsymbol{v}^T \mathbf{W} \boldsymbol{h} - \boldsymbol{b}^T \boldsymbol{v} - \boldsymbol{c}^T \boldsymbol{h} = -\sum_{i=1}^{n} \sum_{j=1}^{m} w_{ij} h_i v_j - \sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i, \tag{3.2}$$

where w_{ij} are real-valued weights associated to the links between units V_j and H_i and b_j and c_i

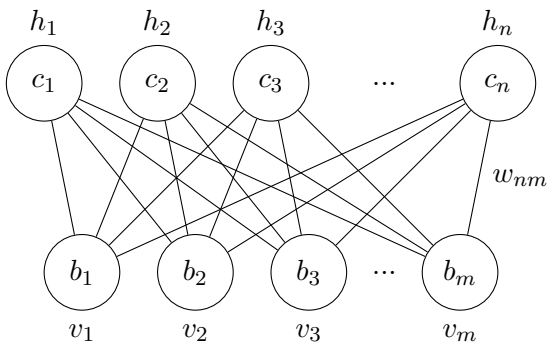


Figure 3.1: Graph structure of a Restricted Boltzmann Machine with m visible units and n hidden units.

are real-valued bias terms associated with V_j and H_i, respectively.

Specifying a generative model with this bipartite interaction structure has two major advantages [25]: (i) it enables capturing both pairwise and higher-order correlations between the visible units, as we will see in Section 3.2, and (ii) it makes it easier to sample from the model using an MCMC method known as block Gibbs sampling, see Section 3.4, which in turn makes the model easier to train.

The partition function Z in Equation (3.1) is given by summing over all possible pairs of visible and hidden vectors:

$$Z = \sum_{\boldsymbol{v}} \sum_{\boldsymbol{h}} \exp(-E(\boldsymbol{v}, \boldsymbol{h})). \tag{3.3}$$

From this definition it is clear that the naive method of computing Z summing over all states could be computationally intractable, unless we find a well-designed algorithm that is able to exploit regularities in the probability distribution to compute Z faster. In the case of RBMs, Long and Servedio formally proved that the partition function Z is intractable [23].

Theorem 3 (Long and Servedio)**.** *There is a universal constant $\epsilon > 0$ such that if $P \neq NP$, then there is no polynomial-time algorithm with the following property: given as input an $n \times n$ matrix A satisfying $\|A\|_\infty \leq \psi(n)$ (where the function ψ grows faster than linearly), the algorithm approximates the partition function Z to within a multiplicative factor of $e^{\epsilon\psi(n)}$.*

The intractable partition function Z implies that the normalized joint probability distribution $p(\boldsymbol{v})$ cannot be evaluated exactly. Therefore, approximations are needed.

3.2 Correlations

Before discussing training, it is worth better understanding the kind of correlations that can be captured using an RBM. We consider a binary-binary RBM, but the following calculations hold in general for all types of RBM [25]. The probability that the network assigns to a visible vector $\boldsymbol{v}$ is given by summing over all possible hidden vectors:

$$p(\boldsymbol{v}) = \sum_{\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) = \sum_{\boldsymbol{h}} \frac{e^{-E(\boldsymbol{v},\boldsymbol{h})}}{Z}. \tag{3.4}$$

But we can also define the marginal energy as

$$p(\boldsymbol{v}) = \frac{e^{-E(\boldsymbol{v})}}{Z}. \tag{3.5}$$

Combining these equations we get

$$E(\boldsymbol{v}) = -\ln \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} = -\sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} \ln \sum_{h_i} e^{c_i h_i + \sum_{j=1}^{m} w_{ij} v_j h_i}. \tag{3.6}$$

Let us introduce the distribution of the hidden units

$$q_i(h_i) = \frac{e^{c_i h_i}}{Z}, \tag{3.7}$$

and the cumulant generating function

$$K_i(t) = \ln \sum_{h_i} q_i(h_i) e^{t h_i} = \sum_r \kappa_i^{(r)} \frac{t^r}{r!}, \tag{3.8}$$

where $\kappa_i^{(r)} = \partial_t^r K_i|_{t=0}$ is the r-th cumulant. Then the marginal energy for the visible units can be re-written as

$$\begin{aligned} E(\boldsymbol{v}) &= -\sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} K_i \left(\sum_{j=1}^{m} w_{ij} v_j \right) \\ &= -\sum_{j=1}^{m} b_j v_j - \sum_{i=1}^{n} \sum_{r=1} \kappa_i^{(r)} \frac{\left(\sum_{j=1}^{m} w_{ij} v_j \right)^r}{r!} \\ &= -\sum_{j=1}^{m} b_j v_j - \sum_{j=1}^{m} \left(\sum_{i=1}^{n} \kappa_i^{(1)} w_{ij} \right) v_j - \frac{1}{2} \sum_{j,l=1}^{m} \left(\sum_{i=1}^{n} \kappa_i^{(2)} w_{ij} w_{il} \right) v_j v_l + \dots \end{aligned} \tag{3.9}$$

Thus, the marginal energy includes all orders of interactions between visible units, weighted by the corresponding cumulant. This is the reason why RBMs have such extraordinary representational power: each hidden unit can encode interactions of arbitrarily high order. Therefore, by combining many different hidden units, we can encode very complex interactions. Another strength of such models is that they are able to learn which orders of interactions are important directly from the data, without the need for a prior specification, like in MaxEnt models.

3.3 Conditional Distributions

The bipartite graph structure of the RBM has the special property of not having intra-layer connections. Mathematically, this means that its conditional distributions $p(\boldsymbol{h}|\boldsymbol{v})$ and $p(\boldsymbol{v}|\boldsymbol{h})$ factorize:

$$p(\boldsymbol{h}|\boldsymbol{v}) = \prod_{i=1}^{n} p(h_i, \boldsymbol{v}), \qquad p(\boldsymbol{v}|\boldsymbol{h}) = \prod_{j=1}^{m} p(v_j, \boldsymbol{h}). \tag{3.10}$$

Let us find the expressions for these factors [13]. First, let $\boldsymbol{v}_{-l}$ denote the state of all visible units except the l-th one and define

$$\alpha_l(\boldsymbol{h}) = -\sum_{i=1}^{n} w_{il} h_i - b_l, \tag{3.11}$$

and

$$\beta(\boldsymbol{v}_{-l}, \boldsymbol{h}) = -\sum_{i=1}^{n} \sum_{j=1, j\neq l}^{m} w_{ij} v_j h_i - \sum_{j=1, j\neq l}^{m} b_j v_j - \sum_{i=1}^{n} c_i h_i, \tag{3.12}$$

so that the energy can be written as

$$E(\boldsymbol{v}, \boldsymbol{h}) = \beta(\boldsymbol{v}_{-l}, \boldsymbol{h}) + v_l \alpha_l(\boldsymbol{h}). \tag{3.13}$$

Then, we have

$$\begin{aligned}
p(V_l = 1|\boldsymbol{h}) &= p(V_l = 1|\boldsymbol{v}_{-l}, \boldsymbol{h}) = \frac{p(V_l = 1, \boldsymbol{v}_{-l}, \boldsymbol{h})}{p(\boldsymbol{v}_{-l}, \boldsymbol{h})} \\
&= \frac{e^{-E(v_l=1, \boldsymbol{v}_{-l}, \boldsymbol{h})}}{e^{-E(v_l=1, \boldsymbol{v}_{-l}, \boldsymbol{h})} + e^{-E(v_l=0, \boldsymbol{v}_{-l}, \boldsymbol{h})}} = \frac{e^{-\beta(\boldsymbol{v}_{-l}, \boldsymbol{h}) - 1 \cdot \alpha_l(\boldsymbol{h})}}{e^{-\beta(\boldsymbol{v}_{-l}, \boldsymbol{h}) - 1 \cdot \alpha_l(\boldsymbol{h})} + e^{-\beta(\boldsymbol{v}_{-l}, \boldsymbol{h}) - 0 \cdot \alpha_l(\boldsymbol{h})}} \\
&= \frac{e^{-\beta(\boldsymbol{v}_{-l}, \boldsymbol{h})} e^{-\alpha_l(\boldsymbol{h})}}{e^{-\beta(\boldsymbol{v}_{-l}, \boldsymbol{h})} e^{-\alpha_l(\boldsymbol{h})} + e^{-\beta(\boldsymbol{v}_{-l}, \boldsymbol{h})}} = \frac{e^{-\beta(\boldsymbol{v}_{-l}, \boldsymbol{h})} e^{-\alpha_l(\boldsymbol{h})}}{e^{-\beta(\boldsymbol{v}_{-l}, \boldsymbol{h})} (e^{-\alpha_l(\boldsymbol{h})} + 1)} \\
&= \frac{e^{-\alpha_l(\boldsymbol{h})}}{e^{-\alpha_l(\boldsymbol{h})} + 1} = \sigma(-\alpha_l(\boldsymbol{h})) \\
&= \sigma\left(\sum_{i=1}^{n} w_{il} h_i + b_l \right),
\end{aligned} \tag{3.14}$$

where $\sigma(x) = 1/(1+e^{-x})$ is the *sigmoid function*. With analogous calculations we can also show that

$$p(H_l = 1|\boldsymbol{v}) = \sigma\left(\sum_{j=1}^{m} w_{lj} v_j + c_l \right). \tag{3.15}$$

The sigmoid function is one of the main activation functions of artificial neurons. Thus, an RBM can be interpreted as a stochastic neural network where the nodes correspond to neurons and edges correspond to synaptic connections. Therefore, the conditional probability of a single variable being one can be interpreted as the firing rate of a stochastic neuron with sigmoid activation function.

Now, let us derive the form of the RBM distribution over $\boldsymbol{V}$. This is done by marginalizing, as we see in Equation (2.12):

$$\begin{aligned}
p(\boldsymbol{v}) &= \sum_{\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v}, \boldsymbol{h})} \\
&= \frac{1}{Z} \sum_{h_1} \sum_{h_2} \cdots \sum_{h_n} e^{\sum_{j=1}^{m} b_j v_j} \prod_{i=1}^{n} e^{h_i \left(c_i + \sum_{j=1}^{m} w_{ij} v_j \right)} \\
&= \frac{1}{Z} e^{\sum_{j=1}^{m} b_j v_j} \sum_{h_1} e^{h_1 \left(c_1 + \sum_{j=1}^{m} w_{1j} v_j \right)} \sum_{h_2} e^{h_2 \left(c_2 + \sum_{j=1}^{m} w_{2j} v_j \right)} \cdots \sum_{h_n} e^{h_n \left(c_n + \sum_{j=1}^{m} w_{nj} v_j \right)}
\end{aligned}$$

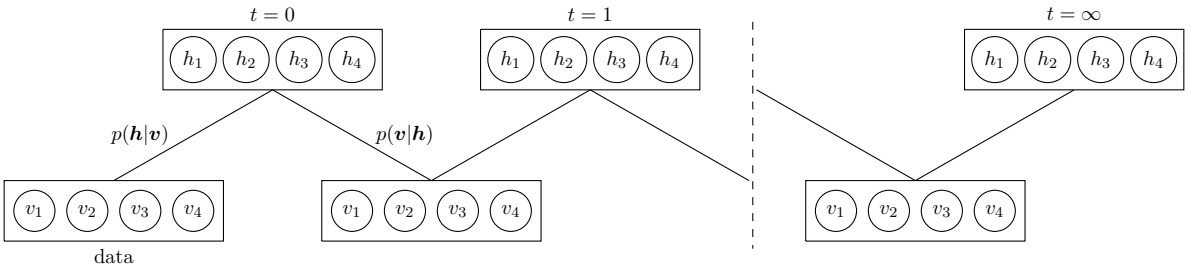


Figure 3.2: Example of block Gibbs sampling for an RBM with four visible units and four hidden units. Here t indicates the number of steps in the chain.

$$
\begin{aligned}
&= \frac{1}{Z} e^{\sum_{j=1}^{m} b_j v_j} \prod_{i=1}^{n} \sum_{h_i} e^{h_i \left(c_i + \sum_{j=1}^{m} w_{ij} v_j\right)} \\
&= \frac{1}{Z} \prod_{j=1}^{m} e^{b_j v_j} \prod_{i=1}^{n} \left(1 + e^{c_i + \sum_{j=1}^{m} w_{ij} v_j}\right). \qquad (3.16)
\end{aligned}
$$

This result shows why an RBM can be regarded as a "product of experts" model, i.e. a model in which multiple "experts" for the individual components of the observations are combined.

3.4 Block Gibbs sampling

The absence of intra-layer connections, which result in the conditional independence between the variables in the same layer, makes Gibbs sampling particularly efficient. Instead of sampling new values for all variables subsequently, the states of all variables in one layer can be sampled jointly, in parallel. Thus, Gibbs sampling results in two steps:

1. sampling a new state $\boldsymbol{h}$ for the hidden units, based on $p(\boldsymbol{h}|\boldsymbol{v})$;
2. sampling a state $\boldsymbol{v}$ for the visible units, based on $p(\boldsymbol{v}|\boldsymbol{h})$.

This is referred to as *block Gibbs sampling* [13, 16]. A schema is shown in Figure 3.2.

3.5 The gradient of the log-likelihood

Equation (2.14) shows that the gradient of a MRF log-likelihood is given by the sum of two terms: the expectation of the energy gradient over the conditional distribution of the hidden units given a training sample $\boldsymbol{v}$, and the expectation of the energy gradient under the RBM distribution [18]. The first term can be efficiently computed, since it factorizes [13]. We have

$$
\begin{aligned}
-\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial w_{ij}} &= \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) h_i v_j \\
&= \sum_{\boldsymbol{h}} \prod_{k=1}^{n} p(h_k|\boldsymbol{v}) h_i v_j \\
&= \sum_{h_i} \sum_{\boldsymbol{h}_{-i}} p(h_i|\boldsymbol{v}) p(\boldsymbol{h}_{-i}|\boldsymbol{v}) h_i v_j
\end{aligned}
$$

$$= \sum_{h_i} p(h_i|\boldsymbol{v}) h_i v_j \underbrace{\sum_{\boldsymbol{h}_{-i}} p(\boldsymbol{h}_{-i}|\boldsymbol{v})}_{=1}$$

$$= p(H_i = 1|\boldsymbol{v}) v_j = \sigma\left(\sum_{j=1}^{m} w_{ij} v_j + c_i\right) v_j. \tag{3.17}$$

The computation of the expectation of the energy gradient under the RBM distribution is intractable for most RBMs because its complexity is exponential in the size the the smallest layer between the visible and the hidden one. In fact, recalling that $p(\boldsymbol{v}, \boldsymbol{h}) = p(\boldsymbol{v})p(\boldsymbol{h}|\boldsymbol{v}) = p(\boldsymbol{h})p(\boldsymbol{v}|\boldsymbol{h})$, we can write

$$\sum_{\boldsymbol{v},\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) \frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} = \begin{cases} \sum_{\boldsymbol{v}} p(\boldsymbol{v}) \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} \\ \sum_{\boldsymbol{h}} p(\boldsymbol{h}) \sum_{\boldsymbol{v}} p(\boldsymbol{v}|\boldsymbol{h}) \frac{\partial E(\boldsymbol{v},\boldsymbol{h})}{\partial \boldsymbol{\theta}} \end{cases}, \tag{3.18}$$

and note that the outer sum runs over 2^m or 2^n terms. Therefore, as discussed in Section 2.3.2, we approximate this expectation by using MCMC techniques. Using Equation (3.17) we can compute the derivative of the log-likelihood of a single training pattern $\boldsymbol{v}$ with respect to w_{ij}:

$$\begin{aligned} \frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial w_{ij}} &= -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) \frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial w_{ij}} + \sum_{\boldsymbol{v},\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) \frac{\partial E(\boldsymbol{v}, \boldsymbol{h})}{\partial w_{ij}} \\ &= \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) h_i v_j - \sum_{\boldsymbol{v}} p(\boldsymbol{v}) \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}) h_i v_j \\ &= p(H_i = 1|\boldsymbol{v}) v_j - \sum_{\boldsymbol{v}} p(\boldsymbol{v}) p(H_i = 1|\boldsymbol{v}) v_j. \end{aligned} \tag{3.19}$$

Then we can compute the mean over the training set $S = \{\boldsymbol{v}_1, \ldots, \boldsymbol{v}_l\}$:

$$\begin{aligned} \frac{1}{l} \sum_{\boldsymbol{v} \in S} \frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial w_{ij}} &= \frac{1}{l} \sum_{\boldsymbol{v} \in S} \left(-\mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v})} \left[\frac{\partial E(\boldsymbol{h}, \boldsymbol{v})}{\partial w_{ij}} \right] + \mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h})} \left[\frac{\partial E(\boldsymbol{h}, \boldsymbol{v})}{\partial w_{ij}} \right] \right) \\ &= \frac{1}{l} \sum_{\boldsymbol{v} \in S} \left(-\mathbb{E}_{p(\boldsymbol{h}|\boldsymbol{v})} \left[v_j h_i \right] - \mathbb{E}_{p(\boldsymbol{v},\boldsymbol{h})} \left[v_j h_i \right] \right) \\ &= \langle v_j h_i \rangle_{p(\boldsymbol{h}|\boldsymbol{v}) q(\boldsymbol{v})} - \langle v_j h_i \rangle_{p(\boldsymbol{v},\boldsymbol{h})} \\ &= \langle v_j h_i \rangle_d - \langle v_j h_i \rangle_m , \end{aligned} \tag{3.20}$$

where q denotes the data distribution, $\langle \cdot \rangle_d$ indicates the expectation under the data, i.e. under $p(\boldsymbol{h}|\boldsymbol{v})p(\boldsymbol{v})$, and $\langle \cdot \rangle_m$ indicates the expectation under the model, i.e. under $p(\boldsymbol{v}, \boldsymbol{h})$. With analogous calculations we can also get the derivatives with respect to the visible and hidden biases:

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial b_j} = v_j - \sum_{\boldsymbol{v}} p(\boldsymbol{v}) v_j, \tag{3.21}$$

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial c_i} = p(H_i = 1|\boldsymbol{v}) - \sum_{\boldsymbol{v}} p(\boldsymbol{v}) p(H_i = 1|\boldsymbol{v}), \tag{3.22}$$

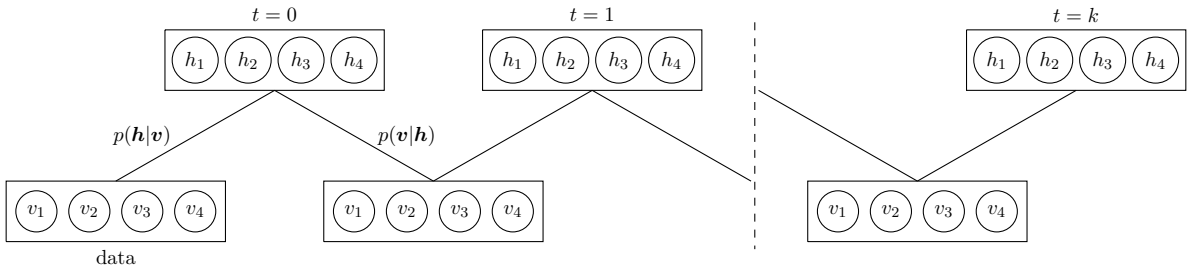


Figure 3.3: Contrastive divergence approximately samples from the model distribution by terminating the Gibbs sampling after k steps, starting from the data.

so that

$$\frac{1}{l}\sum_{\boldsymbol{v}\in S}\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial b_j} = \langle v_j \rangle_d - \langle v_j \rangle_m \,, \tag{3.23}$$

$$\frac{1}{l}\sum_{\boldsymbol{v}\in S}\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial c_i} = \langle h_i \rangle_d - \langle h_i \rangle_m \,. \tag{3.24}$$

Getting unbiased samples of $\langle v_j h_i \rangle_m$, $\langle v_j \rangle_m$ and $\langle h_i \rangle_m$ is difficult. These can be obtained by performing alternating Gibbs sampling for a very long time, making learning very slow. In the next section we discuss much faster learning procedures.

3.6 Training Restricted Boltzmann Machines

RBMs are trained using Maximum Likelihood Estimation (MLE), in which the log-likelihood gradient is approximated using Gibbs sampling and gradient ascent is performed on this approximation. The most common techniques, which will be discussed in this section, are contrastive divergence, persistent contrastive divergence and parallel tempering.

3.6.1 Contrastive Divergence

One drawback of Gibbs sampling is that it may take many back and forth iterations to draw an independent sample. For this reason, called Contrastive Divergence (CD) was introduced as an approximate Gibbs sampling technique [9]. In CD-k, we just perform k iterations of block Gibbs sampling, with k often taken to be as small as 1.

The Gibbs chain is initialized with a training example $\boldsymbol{v}^{(0)}$ and is terminated after k steps, yielding a sample $\boldsymbol{v}^{(k)}$. Each step t consists of a step of block Gibbs sampling, in which $\boldsymbol{h}^{(t)}$ is sampled from $p(\boldsymbol{h}|\boldsymbol{v}^{(t)})$ (positive phase) and then $\boldsymbol{v}^{(t+1)}$ is sampled from $p(\boldsymbol{v}|\boldsymbol{h}^{(t)})$ (negative phase), see Figure 3.3. The gradient of the log-likelihood with respect to a parameter $\boldsymbol{\theta}$ for the training sample $\boldsymbol{v}^{(0)}$ is then approximated by the following expression:

$$\mathrm{CD}_k(\boldsymbol{\theta}, \boldsymbol{v}^{(0)}) = -\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}^{(0)})\frac{\partial E(\boldsymbol{v}^{(0)}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} + \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}^{(k)})\frac{\partial E(\boldsymbol{v}^{(k)}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \tag{3.25}$$

Clearly, this comes at a price. Truncating the Gibbs sampler prevents sampling far away from the starting point, which for CD-k are the data points in the batch. In other words, the approximation given by Equation 3.25 is biased. Therefore, our generative model will be much more accurate around regions of feature space close to our training data. Thus, as is often the case in machine learning, CD-k sacrifices the ability to generalize to some extent in order to make the model easier to train. The following theorem gives a good understanding of the CD approximation and the corresponding bias by showing that the log-likelihood gradient can be expressed as a sum of terms containing the k-th sample [2].

Theorem 4 (Bengio and Delalleau)**.** *For a converging Gibbs chain*

$$\boldsymbol{v}^{(0)} \Rightarrow \boldsymbol{h}^{(0)} \Rightarrow \boldsymbol{v}^{(1)} \Rightarrow \boldsymbol{h}^{(1)} \Rightarrow \dots$$

starting at data point $\boldsymbol{v}^{(0)}$, the log-likelihood gradient can be written as

$$\begin{aligned}\frac{\partial \ln p(\boldsymbol{v}^{(0)})}{\partial \boldsymbol{\theta}} = &- \sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}^{(0)}) \frac{\partial E(\boldsymbol{v}^{(0)}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} + \\ &+ \mathbb{E}_{p(\boldsymbol{v}^{(k)}|\boldsymbol{v}^{(0)})} \left[\sum_{\boldsymbol{h}} p(\boldsymbol{h}|\boldsymbol{v}^{(k)}) \frac{\partial E(\boldsymbol{v}^{(k)}, \boldsymbol{h})}{\partial \boldsymbol{\theta}} \right] + \mathbb{E}_{p(\boldsymbol{v}^{(k)}|\boldsymbol{v}^{(0)})} \left[\frac{\partial \ln p(\boldsymbol{v}^{(k)})}{\partial \boldsymbol{\theta}} \right] \end{aligned} \tag{3.26}$$

and the final term, i.e. the bias, converges to zero as k goes to infinity.

The approximation error depends on the number k of sampling steps as well as on the rate of convergence or the mixing rate of the Gibbs chain. This rate describes how fast the Markov chain approaches the equilibrium distribution and is determined by the transition probabilities of the chain [13]. The mixing rate of the Gibbs chain of an RBM depends on the magnitude of the model parameters [9]. In fact, recall that the conditional probabilities $p(v_j|\boldsymbol{h})$ and $p(h_i|\boldsymbol{v})$ are given by activating $\sum_{i=1}^{n} w_{ij} h_i + b_j$ and $\sum_{j=1}^{m} w_{ij} v_j + c_i$ using the sigmoid function. Therefore, if the absolute values of the parameters are high, the sigmoid function saturates and the conditional probabilities get close to one or zero. When this happens, the states of the Gibbs chain get more and more "predictable", and thus the equilibrium distribution is more difficult to reach.

An upper bound on the expectation of the CD approximation error is given by the following theorem [11].

Theorem 5 (Fischer and Igel)**.** *Let p denote the marginal distribution of the visible units of an RBM and let q be the empirical distribution defined by a set of samples $\boldsymbol{v}_1, \dots, \boldsymbol{v}_l$. Then an upper bound on the expectation of the error of the CD-k approximation of the log-likelihood derivative with respect to some RBM parameter θ_a is given by*

$$\left| \mathbb{E}_{q(\boldsymbol{v}^{(0)})} \left[\mathbb{E}_{p(\boldsymbol{v}^{(k)}|\boldsymbol{v}^{(0)})} \left[\frac{\partial \ln p(\boldsymbol{v}^{(k)})}{\partial \boldsymbol{\theta}} \right] \right] \right| \leq \frac{1}{2} |q - p| \left(1 - e^{-(m+n)\Delta} \right)^k \tag{3.27}$$

Algorithm 1 Contrastive divergence with k steps [13].

Input: RBM, training batch S
Output: Δw_{ij}, Δb_j, Δc_i for $i = 1, \ldots, n$, $j = 1, \ldots, m$

```
Set Δw_ij = Δb_j = Δc_i = 0 for i = 1,...,n, j = 1,...,m
for all v ∈ S do
    v^(0) ← v
    for t = 0,...,k − 1 do
        for i = 1,...,n do
            sample h_i^(t) ~ p(h_i|v^(t))
        for j = 1,...,m do
            sample v_j^(t+1) ~ p(v_j|h^(t))
    for i = 1,...,n, j = 1,...,m do
        Δw_ij ← Δw_ij + p(H_i = 1|v^(0)) v_j^(0) − p(H_i = 1|v^(k)) v_j^(k)
    for j = 1,...,m do
        Δb_j ← Δb_j + v_j^(0) − v_j^(k)
    for i = 1,...,n do
        Δc_i ← Δc_i + p(H_i = 1|v^(0)) − p(H_i = 1|v^(k))
```

with

$$\Delta = \max\left\{ \max_{l \in \{1,\ldots,m\}} \vartheta_l, \max_{l \in \{1,\ldots,n\}} \xi_l \right\} \tag{3.28}$$

where

$$\vartheta_l = \max\left\{ \left| \sum_{i=1}^{n} 1_{w_{il}>0} w_{il} + b_l \right|, \left| \sum_{i=1}^{n} 1_{w_{il}<0} w_{il} + b_l \right| \right\} \tag{3.29}$$

and

$$\xi_l = \max\left\{ \left| \sum_{j=1}^{m} 1_{w_{lj}>0} w_{lj} + c_l \right|, \left| \sum_{j=1}^{m} 1_{w_{lj}<0} w_{lj} + c_l \right| \right\} \tag{3.30}$$

Note that the bound on the error depends on the absolute values of the RBM parameters, on the size of the RBM, and on the distance in variation between the modeled distribution and the starting distribution of the Gibbs chain.

The bias can lead to a distortion of the learning process: after a certain number of iterations the likelihood can start to diverge, in the sense that it systematically decreases if the number of sampling steps k is not large enough [12]. This is a severe problem because the log-likelihood is not tractable for most RBMs, and therefore this misbehavior can not be displayed and used as a stopping criterion. Since the bias depends on the magnitude of the weights, weight decay can help to prevent this problem. However, the weight decay parameter λ, see equation (2.11), is difficult to tune. If it is too small, weight decay has no effect. If it is too large, learning

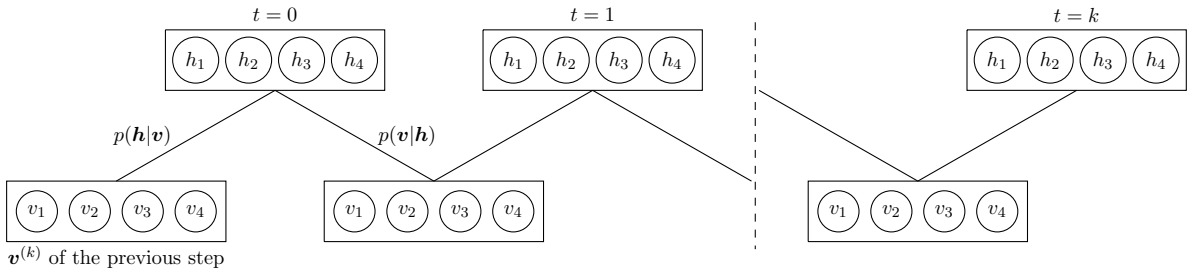


Figure 3.4: Persistent contrastive divergence approximately samples from the model distribution by terminating the Gibbs sampling after k steps, starting from the visible state of the previous update step.

converges to models with low likelihood (see Section 6.2.1) [12].

A batch version of CD-k is shown in Algorithm 1. In this case, in each step the gradient is computed using the complete training set. This is referred to as *batch learning*. However, when dealing with large datasets it is often more efficient to use only a subset $S' \subset S$ – called mini-batch – in every iteration, reducing the computational burden. This is referred to as *online learning*.

3.6.2 Persistent Contrastive Divergence

In Persistent Contrastive Divergence (PCD) [36], rather than restarting the Gibbs sampler from the data at each step – which is the essence of CD – Gibbs sampling starts from the visible state in the previous step. In other words, we keep "persistent" chains that are run for k Gibbs steps after each parameter update, in which the initial state of the current Gibbs chain is equal to $\boldsymbol{v}^{(k)}$ from the previous update step. A schema is shown in Figure 3.4. The idea behind PCD is that, if the learning rate is sufficiently small, one could assume that the chains stay close to the stationary distribution and thus the model parameters evolve slowly. The number of persistent chains used for sampling is a hyper parameter of the algorithm. In the canonical form, there exists one Markov chain per training example in a batch.

There also exists a variant of PCD called Fast Persistent Contrastive Divergence (FPCD), which tries to reach a faster mixing of the Gibbs chain by introducing additional parameters w_{ij}^f, b_j^f, c_i^f – referred to as *fast parameters* – to the conditional distributions used for Gibbs sampling:

$$p(H_i = 1|\boldsymbol{v}) = \sigma\left(\sum_{j=1}^{m}(w_{ij} + w_{ij}^f)v_j + (c_i + c_i^f)\right), \tag{3.31}$$

$$p(V_j = 1|\boldsymbol{h}) = \sigma\left(\sum_{i=1}^{n}(w_{ij} + w_{ij}^f)h_i + (b_j + b_j^f)\right). \tag{3.32}$$

The learning procedure is the same as for PCD, but it requires larger learning rates and the weight decay parameter to change faster. However, this variant seems not to lead to major improvements [12].

3.6.3 Parallel tempering

In Parallel Tempering (PT) we introduce multiple Gibbs chains that sample from more and more smoothed replicas of the original distribution [10, 31]. Given an ordered set of M temperatures $1 = T_1 < T_2 < \cdots < T_M$, we define a set of M Markov chains with stationary distributions

$$p_r(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z_r} \exp\left(-\frac{E(\boldsymbol{v}, \boldsymbol{h})}{T_r}\right), \qquad r = 1, \ldots, M \tag{3.33}$$

where $Z_r = \sum_{\boldsymbol{v},\boldsymbol{h}} e^{-\frac{E(\boldsymbol{v},\boldsymbol{h})}{T_r}}$. Note that p_1 is the model distribution. Chains with higher temperatures have more distributed probability density and therefore their mixing rate is larger. In particular, for $T \to \infty$ we get the uniform distribution, where the samples are independent and the stationary distribution is reached immediately.

Algorithm 2 Parallel tempering with k steps and M Markov chains [13].

Input: RBM, minibatch S
Output: Δw_{ij}, Δb_j, Δc_i for $i = 1, \ldots, n$, $j = 1, \ldots, m$

Set $\Delta w_{ij} = \Delta b_j = \Delta c_i = 0$ for $i = 1, \ldots, n$, $j = 1, \ldots, m$
for all $\boldsymbol{v} \in S$ **do**
  for $r = 1, \ldots, M$ **do**
    $\boldsymbol{v}_r^{(0)} \leftarrow \boldsymbol{v}_r$
    for $i = 1, \ldots, n$ **do**
      sample $h_{r,i}^{(0)} \sim p(h_{r,i} | \boldsymbol{v}_r^{(0)})$
    for $t = 0, \ldots, k-1$ **do**
      for $j = 1, \ldots, m$ **do**
        sample $v_{r,j}^{(t+1)} \sim p(v_{r,j} | \boldsymbol{h}_r^{(t)})$
      for $i = 1, \ldots, n$ **do**
        sample $h_{r,i}^{(t+1)} \sim p(h_{r,i} | \boldsymbol{v}_r^{(t+1)})$
    $\boldsymbol{v}_r \leftarrow \boldsymbol{v}_r^{(k)}$
  for $r \in \{s | 2 \leq s \leq M \text{ and } s \mod 2 = 0\}$ **do**
    swap $(\boldsymbol{v}_r^{(k)}, \boldsymbol{h}_r^{(k)})$ and $(\boldsymbol{v}_{r-1}^{(k)}, \boldsymbol{h}_{r-1}^{(k)})$ with probability given by (3.35)
  for $r \in \{s | 3 \leq s \leq M \text{ and } s \mod 2 = 1\}$ **do**
    swap $(\boldsymbol{v}_r^{(k)}, \boldsymbol{h}_r^{(k)})$ and $(\boldsymbol{v}_{r-1}^{(k)}, \boldsymbol{h}_{r-1}^{(k)})$ with probability given by (3.35)
  for $i = 1, \ldots, n$, $j = 1, \ldots, m$ **do**
    $\Delta w_{ij} \leftarrow \Delta w_{ij} + p(H_i = 1 | \boldsymbol{v}) v_j - p(H_i = 1 | \boldsymbol{v}_1^{(k)}) v_{1,j}^{(k)}$
  for $j = 1, \ldots, m$ **do**
    $\Delta b_j \leftarrow \Delta b_j + v_j - v_{1,j}^{(k)}$
  for $i = 1, \ldots, n$ **do**
    $\Delta c_i \leftarrow \Delta c_i + p(H_i = 1 | \boldsymbol{v}) - p(H_i = 1 | \boldsymbol{v}_1^{(k)})$

Algorithm 2 shows the pseudo-code of PT. In each step of the algorithm we run k Gibbs sampling steps in each Markov chain, obtaining the samples $(\boldsymbol{v}_1^{(k)}, \boldsymbol{h}_1^{(k)}), \ldots, (\boldsymbol{v}_M^{(k)}, \boldsymbol{h}_M^{(k)})$. Then, two neighbouring Gibbs chains T_r and T_{r-1} may exchange particles $(\boldsymbol{v}_r^{(k)}, \boldsymbol{h}_r^{(k)})$ and $(\boldsymbol{v}_{r-1}^{(k)}, \boldsymbol{h}_{r-1}^{(k)})$ with

probability based on the Metropolis ratio

$$\omega = \min\left\{1, \frac{p_r\left(\boldsymbol{v}_{r-1}^{(k)}, \boldsymbol{h}_{r-1}^{(k)}\right) p_{r-1}\left(\boldsymbol{v}_r^{(k)}, \boldsymbol{h}_r^{(k)}\right)}{p_r\left(\boldsymbol{v}_r^{(k)}, \boldsymbol{h}_r^{(k)}\right) p_{r-1}\left(\boldsymbol{v}_{r-1}^{(k)}, \boldsymbol{h}_{r-1}^{(k)}\right)}\right\}, \tag{3.34}$$

which for the RBM becomes

$$\omega = \min\left\{1, \exp\left(\left(\frac{1}{T_r} - \frac{1}{T_{r-1}}\right)\left(E\left(\boldsymbol{v}_r^{(k)}, \boldsymbol{h}_r^{(k)}\right) - E\left(\boldsymbol{v}_{r-1}^{(k)}, \boldsymbol{h}_{r-1}^{(k)}\right)\right)\right)\right\}. \tag{3.35}$$

If $\omega > r$, where $r \sim U(0,1)$ is an uniformly distributed random number, the swap is done. After these swaps we take $\boldsymbol{v}_1$ as a sample from the RBM distribution. This procedure is repeated L times, yielding the samples $v_{1,1}, \ldots, v_{1,L}$ which are used for the approximation of the expected value under the model distribution in the log-likelihood gradient. Usually, L is set to the number of samples in the mini-batch of training data.

Compared to CD and PCD, PT introduces computational overhead but leads to more mixed Markov chains and thus less biased gradient approximations.

3.7 Practical Considerations

In this chapter we reviewed the main aspects of training RBMs. However, it is impossible to provide an exhaustive guide that deals with all the possible parameters and procedures. A brief summary of useful tricks and heuristics has been compiled by Geoffrey Hinton [18]. Here we report the key points:

Size of the mini-batches. For datasets that contain a small number of equiprobable classes, the ideal mini-batch size is often equal to the number of classes. Moreover, each mini-batch should ideally contain one example of each class to reduce the sampling error when estimating the gradient for the whole training set. For other datasets, first randomize the order of the training example and then use mini-batches of size about 10.

Initialization of the parameters. Make sure that the weights have random initial values to break the symmetry. Hinton suggests taking the weights w_{ij} from a Gaussian with mean zero and standard deviation $\sigma = 0.01$. An alternative initialization scheme proposed by Glorot and Bengio [15] instead chooses the standard deviation to scale with the size of the layers: $\sigma = 2/\sqrt{N_v + N_h}$, where N_v and N_h are number of visible and hidden units respectively. The bias of the hidden units is initialized to zero while the bias of the visible units is set to $\ln(p_j/(1-p_j))$ where p_j is the proportion of training vectors in which v_j is on, that is $p_j = \langle v_j \rangle_d$.

Number of hidden units. First, estimate how many bits it would take to describe each data-vector if we were using a good model. Then, multiply that estimate by the number of training cases and use a number of parameters that is about an order of magnitude smaller. If

the training cases are highly redundant, use fewer parameters.

Reconstruction error. It can be used but it should not be completely trusted. One good practice is to visualize the histograms of the weights, visible biases and hidden biases. One can also consider the histogram of the increments to these parameters.

Monitoring overfitting. After every few epochs, compute the average free energy of a representative subset of the training data

$$\langle F(\boldsymbol{v})\rangle = \left\langle -\sum_{j=1}^{m} v_j b_j - \sum_{i=1}^{n} \log\left(1 + e^{\sum_{j=1}^{m} w_{ij} v_j + c_i}\right) \right\rangle \tag{3.36}$$

and compare it with the average free energy of a validation set. If the gap starts growing, the model is overfitting.

Regularization. Use an L1 or L2 penalty, typically only on the weight parameters, not the biases. Values for the weight-cost coefficient for L2 weight decay typically range from 0.01 to 0.00001. Dropout can also decrease overfitting when training with CD and PCD.

Sparsity. We can encourage sparse activities of the binary hidden units by choosing a probability $p \ll 1$ of being active and by enforcing the actual probability of being active q to be close to p. q is estimated by using an exponential decaying average of the mean probability that a unit is active in each mini-batch:

$$q_{new} = \lambda q_{old} + (1 - \lambda) q_{current}, \tag{3.37}$$

where $q_{current}$ is the mean activation of the hidden units on the current mini-batch. The sparsity target p is set to between 0.01 and 0.1, and the decay rate λ is set to between 0.9 and 0.99.

Momentum. Start with $\nu = 0.5$. Once the large initial progress in the reduction of the reconstruction error has settled down to gentle progress, increase $\nu = 0.9$. This shock may cause a transient increase in the reconstruction error. If this causes a more lasting instability, keep reducing the learning rate by factors of 2 until the instability disappears.

Learning Rates. Typically, it is helpful to reduce the learning rate in later stages of training. A good rule of thumb is to look at the histogram of the weight updates and at the histogram of the weights. The updates should be about 10^{-3} times the weights.

Updates for CD-1 and PCD-1. If the visible units are using probabilities instead of binary values, there are two ways to collect the positive statistics for contrastive divergence: $\langle p_j h_i\rangle$ or $\langle p_i p_j\rangle$, where p_i is the probability that h_i takes value 1, which is also the expected value of h_i. Using h_i is closer to the mathematical model of an RBM, but using p_j allows faster learning because it introduces less noise. However, using h_i can create less noise when computing the

difference between positive and negative statistics. Hinton suggests to always use states when the hidden units are being driven by data and to always use probabilities when they are driven by reconstructions. If the visible units use the sigmoid function, it is better to use probabilities both for the data and the reconstructions. Finally, when collecting the statistics for learning weights or biases, use the probabilities.

4 Centered Restricted Boltzmann Machines

In the first decade of the 2000s, when RBMs were one of the main focuses of attention in the deep learning community, two major problems have been reported [26].

The first problem is that the bias of the gradient approximation introduced by using only a few (usually one) steps of Gibbs sampling may lead to a divergence of the log-likelihood during training. As discussed in the previous chapter, this problem is solved by using more advanced techniques that allow a faster mixing of the Gibbs chain, such as parallel tempering.

The second problem is that the learning process is not invariant to the data representation. For example, if we train an RBM on the MNIST dataset and then on the 1-MNIST dataset – obtained by flipping each bit in the MNIST [34] (see Figure 4.1) – we obtain different performances. This is due to missing invariance properties of the gradient with respect to this flip transformation [13]. Such problem was first solved Cho, Raiko and Ilin by introducing the so called *enhanced gradient*, which is derived by calculating a weighted average over the gradients one gets by applying any possible bit flip combination on the data set [6]. Tang and Sutskever found another (simpler) solution, which consists in subtracting the data mean from the visible variables [34]. This is known as the *centering trick*, which was originally proposed for feed forward neural networks [22, 32]. These techniques lead to a model that reaches similar performances and results both on the MNIST and the 1-MNIST dataset. More recently, Montavon and Müller extended the centering trick also to the hidden units [27]. Actually, all these techniques are just particular cases of a more general centering trick, in which an offset is subtracted both from the hidden and visible units.

(a) MNIST. (b) 1-MNIST.

Figure 4.1: MNIST and 1-MNIST datasets examples.

4.1 The centering trick

The centering trick consists in shifting the visible and hidden variables by some offset parameters $\mu = (\mu_1, \ldots, \mu_N)$ and $\lambda = (\lambda_1, \ldots, \lambda_M)$, respectively. The energy for the corresponding centered binary RBM is given by

$$E(\boldsymbol{v}, \boldsymbol{h}) = -(\boldsymbol{v} - \boldsymbol{\mu})^T \boldsymbol{b} - \boldsymbol{c}^T(\boldsymbol{h} - \boldsymbol{\lambda}) - (\boldsymbol{v} - \boldsymbol{\mu})\mathbf{W}(\boldsymbol{h} - \boldsymbol{\lambda}). \tag{4.1}$$

For $\boldsymbol{\mu} = \boldsymbol{\lambda} = \mathbf{0}$ we recover the normal binary RBM, for $\boldsymbol{\mu} = \langle \boldsymbol{v} \rangle_d$ and $\boldsymbol{\lambda} = \mathbf{0}$ we obtain the original centering trick by Tang and Sutskever and for $\boldsymbol{\mu} = \langle \boldsymbol{v} \rangle_d$ and $\boldsymbol{\lambda} = \langle \boldsymbol{h} \rangle_d$ we obtain the model by Montavon and Müller.

The expressions for the factors of the conditional probabilities (3.10) are now given by

$$p(V_l = 1|\boldsymbol{h}) = \sigma\left(\sum_{i=1}^{n} w_{il}(h_i - \lambda_i) + b_l\right), \tag{4.2}$$

$$p(H_l = 1|\boldsymbol{v}) = \sigma\left(\sum_{j=1}^{n} w_{lj}(v_j - \mu_j) + c_l\right). \tag{4.3}$$

Again, we can find the RBM distribution over $\boldsymbol{V}$ by marginalizing (see Equation (2.12)):

$$\begin{aligned}
p(\boldsymbol{v}) &= \sum_{\boldsymbol{h}} p(\boldsymbol{v}, \boldsymbol{h}) = \frac{1}{Z} \sum_{\boldsymbol{h}} e^{-E(\boldsymbol{v},\boldsymbol{h})} \\
&= \frac{1}{Z} \sum_{h_1} \sum_{h_2} \cdots \sum_{h_n} e^{\sum_{j=1}^{m} b_j(v_j - \mu_j)} \prod_{i=1}^{n} e^{(h_i - \lambda_i)\left(c_i + \sum_{j=1}^{m} w_{ij}(v_j - \mu_j)\right)} \\
&= \frac{1}{Z} e^{\sum_{j=1}^{m} b_j(v_j - \mu_j)} \prod_{i=1}^{n} \sum_{h_i} e^{(h_i - \lambda_i)\left(c_i + \sum_{j=1}^{m} w_{ij}(v_j - \mu_j)\right)} \\
&= \frac{1}{Z} \prod_{j=1}^{m} e^{b_j(v_j - \mu_j)} \prod_{i=1}^{n} \left(e^{-\lambda_i\left(c_i + \sum_{j=1}^{m} w_{ij}(v_j - \mu_j)\right)} + e^{(1-\lambda_i)\left(c_i + \sum_{j=1}^{m} w_{ij}(v_j - \mu_j)\right)}\right) \\
&= \frac{1}{Z} \prod_{j=1}^{m} e^{b_j(v_j - \mu_j)} \prod_{i=1}^{n} e^{-\lambda_i\left(c_i + \sum_{j=1}^{m} w_{ij}(v_j - \mu_j)\right)} \left(1 + e^{c_i + \sum_{j=1}^{m} w_{ij}(v_j - \mu_j)}\right).
\end{aligned} \tag{4.4}$$

Finally, the log-likelihood gradients now take the form:

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial w_{ij}} \equiv \Delta w_{ij} = \langle (v_j - \mu_j)(h_i - \lambda_i) \rangle_d - \langle (v_j - \mu_j)(h_i - \lambda_i) \rangle_m \tag{4.5}$$

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial b_j} \equiv \Delta b_j = \langle v_j - \mu_j \rangle_d - \langle v_j - \mu_j \rangle_m = \langle v_j \rangle_d - \langle v_j \rangle_m \tag{4.6}$$

$$\frac{\partial \ln \mathcal{L}(\boldsymbol{\theta}|\boldsymbol{v})}{\partial c_i} \equiv \Delta c_i = \langle h_i - \lambda_i \rangle_d - \langle h_i - \lambda_i \rangle_m = \langle h_i \rangle_d - \langle h_i \rangle_m, \tag{4.7}$$

which in vector form read

$$\Delta \mathbf{W} = \left\langle (\boldsymbol{v} - \boldsymbol{\mu})(\boldsymbol{h} - \boldsymbol{\lambda})^T \right\rangle_d - \left\langle (\boldsymbol{v} - \boldsymbol{\mu})(\boldsymbol{h} - \boldsymbol{\lambda})^T \right\rangle_m \tag{4.8}$$

$$\Delta \boldsymbol{b} = \langle \boldsymbol{v} \rangle_d - \langle \boldsymbol{v} \rangle_m \tag{4.9}$$

$$\Delta \boldsymbol{c} = \langle \boldsymbol{h} \rangle_d - \langle \boldsymbol{h} \rangle_m . \tag{4.10}$$

These equations show that centering only affects the gradient with respect to the weights. The important results is that it can be shown that the gradient of a centered RBM is invariant to flip transformations if a flip of v_j to $1 - v_j$ implies a change of μ_j to $1 - \mu_j$ and a flip of h_i to $1 - h_i$ implies a change of λ_i to $1 - \lambda_i$ [26]. This holds for $\mu_j = \lambda_i = 0.5$ but also for the expectation values of v_j and h_j under any distribution. Moreover, if the offsets are set to the expectation values, centered RBMs are also invariant to any shift of variables, not only to flip transformations [26].

Montavon and Müller have shown that centering improves the conditioning of the underlying optimization problem [27]. More precisely, the ratio between the highest and the lowest eigenvalue of the Hessian matrix is smaller. This ratio is known as *condition number* of the Hessian and it encodes how hard a strongly convex problem is to solve. Larger condition numbers imply slower convergence of gradient descent because in some directions the gradient will change rapidly and in others it will change very slowly.

4.2 Training centered Restricted Boltzmann Machines

If we set $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ to the expected values of the variables, which is the most common choice, these values may depend on the model parameters and thus they may change during training. Therefore, we need to adapt the standard learning algorithm so that the offsets are updated to match the expectations under the new distribution we get after each parameter update. Moreover, when updating the offsets we need to transform the RBM parameters such that the probability distribution remains the same. An RBM with offsets $\boldsymbol{\mu}$ and $\boldsymbol{\lambda}$ can be transformed into an RBM with offsets $\tilde{\boldsymbol{\mu}}$ and $\tilde{\boldsymbol{\lambda}}$ by

$$\tilde{\mathbf{W}} = \mathbf{W}, \tag{4.11}$$

$$\tilde{\boldsymbol{b}} = \boldsymbol{b} + \mathbf{W}(\tilde{\boldsymbol{\lambda}} - \boldsymbol{\lambda}), \tag{4.12}$$

$$\tilde{\boldsymbol{c}} = \boldsymbol{c} + \mathbf{W}^T(\tilde{\boldsymbol{\mu}} - \boldsymbol{\mu}), \tag{4.13}$$

such that $E(\boldsymbol{v}, \boldsymbol{h}|\boldsymbol{\theta}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = E(\boldsymbol{v}, \boldsymbol{h}|\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\mu}}, \tilde{\boldsymbol{\lambda}}) + const$ is guaranteed. Clearly, these equations can be used to transform a centered RBM into a normal one and vice versa, emphasizing that normal and centered RBMs are just different parametrizations of the same model class.

Algorithm 3 shows the pseudo-code to train a centered RBM. Notice that $\langle \cdot \rangle$ denotes the average over the samples of the current batch. Thus, for example, $\langle \boldsymbol{v}_d \rangle$ is the average of the samples $\boldsymbol{v}_d$ in the current batch, which is taken as approximation of $\langle \boldsymbol{v} \rangle_d$. Similarly, $\langle \boldsymbol{h}_d \rangle = \langle p(\boldsymbol{h} = \mathbf{1}|\boldsymbol{v}_d) \rangle$

is an approximation for $\langle \boldsymbol{h} \rangle_d$. Note that the offsets are updated using a moving average with sliding factors $\zeta_\mu, \zeta_\lambda \in (0, 1)$ (usually $\zeta \sim 0.01$). This is done to get a smoother approximation of the parameter updates in case the approximation of the mean values can be biased. For example, if we use the model mean or we have small mini-batch sizes the use of the moving average leads to stabler updates.

Algorithm 3 Training a centered RBM [26].

Input: RBM, data.
Output: Trained RBM.

Initialize $\mathbf{W}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{\mu}$, $\boldsymbol{\lambda}$, η (learning rate), ζ_μ, ζ_λ (moving average factors).
repeat
  for all batch in data **do**
    for all sample $\boldsymbol{v}$ in batch **do**
      Compute $\boldsymbol{h}_d$, $\boldsymbol{v}_m$ and $\boldsymbol{h}_m$ using, for example, PCD or PT;
    Estimate $\boldsymbol{\mu}_{batch} = \langle \boldsymbol{v}_d \rangle$ and $\boldsymbol{\lambda}_{batch} = \langle \boldsymbol{h}_d \rangle$
    `/* Transform the parameters with respect to the new offsets */`
    $\boldsymbol{b} \leftarrow \boldsymbol{b} + \zeta_\lambda \mathbf{W}(\boldsymbol{\lambda}_{batch} - \boldsymbol{\lambda})$;
    $\boldsymbol{c} \leftarrow \boldsymbol{c} + \zeta_\mu \mathbf{W}^T(\boldsymbol{\mu}_{batch} - \boldsymbol{\mu})$;
    `/* Update the offsets using a moving average with factors` ζ_μ `and` ζ_λ `*/`
    $\boldsymbol{\mu} \leftarrow (1 - \zeta_\mu)\boldsymbol{\mu} + \zeta_\mu \boldsymbol{\mu}_{batch}$;
    $\boldsymbol{\lambda} \leftarrow (1 - \zeta_\lambda)\boldsymbol{\lambda} + \zeta_\lambda \boldsymbol{\lambda}_{batch}$;
    `/* Update the parameters according to the gradients */`
    $\Delta \mathbf{W} \leftarrow \left\langle (\boldsymbol{v}_d - \boldsymbol{\mu})(\boldsymbol{h}_d - \boldsymbol{\lambda})^T \right\rangle - \left\langle (\boldsymbol{v}_m - \boldsymbol{\mu})(\boldsymbol{h}_m - \boldsymbol{\lambda})^T \right\rangle$;
    $\Delta \boldsymbol{b} \leftarrow \langle \boldsymbol{v}_d \rangle - \langle \boldsymbol{v}_m \rangle$;
    $\Delta \boldsymbol{c} \leftarrow \langle \boldsymbol{h}_d \rangle - \langle \boldsymbol{h}_m \rangle$;
    $\mathbf{W} \leftarrow \mathbf{W} + \eta \Delta \mathbf{W}$;
    $\boldsymbol{b} \leftarrow \boldsymbol{b} + \eta \Delta \boldsymbol{b}$;
    $\boldsymbol{c} \leftarrow \boldsymbol{c} + \eta \Delta \boldsymbol{c}$;
until training is finished;

4.2.1 Centering the gradient

Instead of centering the parameters, we can also center the gradients, obtaining a "centered parameter update". This leads to the following updates [26]:

$$\Delta_c w_{ij} = \langle (v_j - \mu)(h_i - \lambda_i) \rangle_d - \langle (v_j - \mu_j)(h_i - \lambda_i) \rangle_m \tag{4.14}$$

$$\Delta_c b_j = \langle v_j \rangle_d - \langle v_j \rangle_m - \Delta_c w_{ij} \lambda_i \tag{4.15}$$

$$\Delta_c c_i = \langle h_i \rangle_d - \langle h_i \rangle_m - \Delta_c w_{ij}^T \lambda_j, \tag{4.16}$$

which in vector form read

$$\Delta \mathbf{W} = \left\langle (\boldsymbol{v} - \boldsymbol{\mu})(\boldsymbol{h} - \boldsymbol{\lambda})^T \right\rangle_d - \left\langle (\boldsymbol{v} - \boldsymbol{\mu})(\boldsymbol{h} - \boldsymbol{\lambda})^T \right\rangle_m \tag{4.17}$$

$$\Delta \boldsymbol{b} = \langle \boldsymbol{v} \rangle_d - \langle \boldsymbol{v} \rangle_m - \Delta_c \mathbf{W} \boldsymbol{\lambda} \tag{4.18}$$

$$\Delta \boldsymbol{c} = \langle \boldsymbol{h} \rangle_d - \langle \boldsymbol{h} \rangle_m - \Delta_c \mathbf{W}^T \boldsymbol{\mu}. \tag{4.19}$$

If we set $\boldsymbol{\mu} = (\langle \boldsymbol{v} \rangle_d + \langle \boldsymbol{v} \rangle_m)/2$ and $\boldsymbol{\lambda} = (\langle \boldsymbol{h} \rangle_d + \langle \boldsymbol{h} \rangle_m)/2$ we recover the enhanced gradient [26]. This confirms the fact that the enhanced gradient gradient is just a particular kind of centering trick. The pseudocode for training a normal RBM using the centered gradient is shown in Algorithm 4.

Algorithm 4 Training a normal RBM using the centered gradient [26].

Input: RBM, data.

Output: Trained RBM.

Initialize $\mathbf{W}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{\mu}$, $\boldsymbol{\lambda}$, η (learning rate), ζ_μ, ζ_λ (moving average factors).
repeat
  for all batch in data **do**
    for all sample $\boldsymbol{v}$ in batch **do**
      Compute $\boldsymbol{h}_d$, $\boldsymbol{v}_m$ and $\boldsymbol{h}_m$ using, for example, PCD or PT;
    Estimate $\boldsymbol{\mu}_{batch} = \langle \boldsymbol{v}_d \rangle$ and $\boldsymbol{\lambda}_{batch} = \langle \boldsymbol{h}_d \rangle$
    `/* Update the offsets using a moving average with factors` ζ_μ `and` ζ_λ `*/`
    $\boldsymbol{\mu} \leftarrow (1 - \zeta_\mu)\boldsymbol{\mu} + \zeta_\mu \boldsymbol{\mu}_{batch}$;
    $\boldsymbol{\lambda} \leftarrow (1 - \zeta_\lambda)\boldsymbol{\lambda} + \zeta_\lambda \boldsymbol{\lambda}_{batch}$;
    `/* Update the parameters using the centered gradient */`
    $\Delta_c \mathbf{W} \leftarrow \left\langle (\boldsymbol{v}_d - \boldsymbol{\mu})(\boldsymbol{h}_d - \boldsymbol{\lambda})^T \right\rangle - \left\langle (\boldsymbol{v}_m - \boldsymbol{\mu})(\boldsymbol{h}_m - \boldsymbol{\lambda})^T \right\rangle$;
    $\Delta_c \boldsymbol{b} \leftarrow \langle \boldsymbol{v}_d \rangle - \langle \boldsymbol{v}_m \rangle - \Delta_c \mathbf{W} \boldsymbol{\lambda}$;
    $\Delta_c \boldsymbol{c} \leftarrow \langle \boldsymbol{h}_d \rangle - \langle \boldsymbol{h}_m \rangle - \Delta_c \mathbf{W}^T \boldsymbol{\mu}$;
    $\mathbf{W} \leftarrow \mathbf{W} + \eta \Delta_c \mathbf{W}$;
    $\boldsymbol{b} \leftarrow \boldsymbol{b} + \eta \Delta_c \boldsymbol{b}$;
    $\boldsymbol{c} \leftarrow \boldsymbol{c} + \eta \Delta_c \boldsymbol{c}$;
until training is finished;

5

Restricted Boltzmann Machines with real-valued units

RBMs were initially developed using binary visible and hidden units, for which the probability of being active is given by the sigmoid function, see (3.14) and (3.15). However, there are many other types of units that can be used, in particular when dealing with data that take real values and that are modeled by continuous distributions. In general, it is possible to have continuous valued variables by extending the definition of RBM to a MRF for which the energy is such that Equations (3.10) hold. As follows from Hammersley-Clifford theorem (Theorem 1), this happens for any energy function of the form

$$E(\boldsymbol{v}, \boldsymbol{h}) = \sum_{i=1}^{n}\sum_{j=1}^{m} \psi_{i,j}(h_i, v_j) + \sum_{j=1}^{m} g_j(v_j) + \sum_{i=1}^{n} \mathcal{U}_i(h_i), \tag{5.1}$$

where $\psi_{i,j}$, g_j and $\mathcal{U}_i$ are potentials such that the partition function Z is finite. In this chapter we present Gaussian, binomial and rectified linear units.

5.1 Gaussian units

When dealing with data such as natural images or if we want to represent data with high fidelity we can replace binary visible units by linear units with independent Gaussian noise. The energy function contains a quadratic potential:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} h_i \frac{v_j}{\sigma_j} + \sum_{j=1}^{m} \frac{(v_j - b_j)^2}{2\sigma_j^2} - \sum_{i=1}^{n} c_i h_i, \tag{5.2}$$

where σ_j is the standard deviation for visible unit v_j. One of the main difficulties with this kind of model is learning the variance parameters σ_j, which are constrained to be positive. Therefore, in many applications, each visible unit is normalized to have zero mean and unitary variance. Then, one uses noise-free reconstructions, in which $\sigma_j = 1\ \forall j$, given by the input from the hidden units plus its bias, $\sum_i w_{ij} h_i + b_j$. A different approach was taken by Cho, Ilin and Raiko, who

introduced an energy function which facilitates learning [5]:

$$E(\boldsymbol{v}, \boldsymbol{h}) = -\sum_{i=1}^{n}\sum_{j=1}^{m} w_{ij} h_i \frac{v_j}{\sigma_j^2} + \sum_{j=1}^{m} \frac{(v_j - b_j)^2}{2\sigma_j^2} - \sum_{i=1}^{n} c_i h_i. \tag{5.3}$$

Under this modified energy function, the expressions for the conditional probabilities and the update rules for the parameters contain only σ_j^2. Furthermore, they re-parametrize the variance parameters as $\sigma_j^2 = e^{z_j}$, so that they are naturally constrained to stay positive.

When using Gaussian units, particular attention must be paid to the value of the learning rate. It should be one or two orders of magnitude smaller than the one used with binary units, since there is no upper bound to the components of the reconstruction (i.e. the visible units). Therefore, if one component is very large, its corresponding weights will get a very large update. This makes RBMs with Gaussian visible units less stable with respect to binary ones. If both visible and hidden units are Gaussian, the instability problem becomes even worse. We can also use Gaussian hidden units with binary visible units. In this case $p(\boldsymbol{v}, \boldsymbol{h})$ is Gaussian in the hidden units and $p(\boldsymbol{v})$ can be exactly computed [1]. The result is the equilibrium distribution of a Hopfield model, which is known for modelling only pairwise interactions. However, in general it is well known that non-linear activation functions produce better results. In fact, as we discussed in Section 3.2, non-linear activation functions (e.g. the sigmoid or the ReLU) produce high order (> 2) interactions between units.

5.2 Binomial units

Another way to allow each unit to express more information was introduced by Teh and Hinton when dealing with images of faces, for which binary pixels are far from ideal [35]. They introduced binomial units, which are obtained by making N separate copies of each binary unit, all sharing the same bias and weights. In this way, each pixel can have $N + 1$ different intensities. The nice side effect of shared weights is that all copies receive the same total input and therefore they have the same probability p_i of being active, meaning that we have to perform the computation only once. During the reconstruction of the image from the hidden activities, we can select n replicas to be active with probability $\binom{N}{n} n^{p_i} (N-n)^{1-p_i}$. The expected number of active replicas is Np_i and the variance is $Np_i(1 - p_i)$. For small p_i we get a Poisson distribution and the growth in p_i is exponential in the total input, making learning less stable. Conversely, for $p_i \to 1$ the variance becomes small, which is not desirable. Another nice feature deriving from weights-sharing is that the mathematics underlying binary-binary RBMs remains the same. In particular, the gradient of the log-likelihood (3.20) is unaffected, except that v_j and h_i are now the number of active replicas.

This "replica trick" is a cheap way of simulating an ensemble of neurons. Alternatively, it can be seen as a way of simulating a single neuron over a time interval in which it may produce multiple spikes.

5.3 Rectified linear units

Rectified linear units are a variant of binomial units that brings new interesting features. Let us consider a set of binomial units, where each copy has shared weights and biases. Now, we add different fixed offsets to the biases of duplicated units: the first duplicate bias is offset by -0.5, the second by -1.5, the third by -2.5, and so on, up to $-(N - 0.5)$. Whereas in the binomial case the sum of the duplicated units follows a binomial distribution, here it is possible to prove that the sum of the probabilities of the duplicated units approximately follows a distribution given by

$$\sum_{n=1}^{\infty} \sigma\left(I_i - n + 0.5\right) \approx \log\left(1 + e^{I_i}\right) \tag{5.4}$$

where $I_i = \sum_j w_{ij} v_j + c_i$ [28]. This is a smoothed version of a rectified linear unit function $\text{ReLU}(x) = \max(0, x)$. A nice feature of ReLU units is that, unlike Bernoulli units, they preserve information about the magnitudes of their inputs above threshold. This property is expected for real neurons, in fact ReLU functions were first introduced in the context of theoretical neuroscience [37].

The downside of using a different bias for each copy is that to get the probabilities required for sampling integer values we need to evaluate the logistic function N times. However, Nair and Hinton showed that we can drop the constraint on the rectified linear unit values to be integers and approximate the states as $\max(0, I_i + \mathcal{N}(0, \sigma(I_i)))$, where $\mathcal{N}(0, V)$ is Gaussian noise with zero mean and variance V [28]. An unit that uses this kind of approximation is called *Noisy Rectified Linear Unit* (NReLU), and it has been shown that it works well with CD-1 [28]. However, this sampling heuristic does not suggest the parametric form of the joint binary-NReLU distribution. This means we cannot evaluate it using methods such as Annealed Importance Sampling. In fact, only strictly monotonic activation functions can derive feasible joint and conditional distributions and NReLU is not strictly monotonic [29].

If both visible and hidden units are rectified linear, we may need a smaller learning rate to avoid instability [18]. In fact, if the weight between two rectified linear units is greater than 1 then there is no upper bound to the energy that we can obtain by giving high values to the units. Nevertheless, RBMs composed of rectified linear units are more stable than RBMs composed of Gaussian units because the rectification avoids oscillations between very high positive activity for one mini-batch and very high negative activity for the next one [18].

ReLU units non-linearity allow very good discriminative and generative properties. Nair and Hinton gave an approximate probabilistic interpretation for the $\max(0, I)$ non-linearity [28]. Consider using rectified linear units with zero noise to model data that lies on the surface of an unit hypersphere. Each ReLU unit corresponds to a plane through the centre of the hypersphere and the activity is zero on half of it and on the other half it increases linearly with the distance from that plane. In this way with N units we get 2^N regions on the surface of the hypersphere (which is assumed to be at least N-dimensional). In each of these regions we have different

active units, but their number does not change. Therefore, we have a different linear model in each region.

5.3.1 Emergence of compositional representations

Once the parameters of an RBM are trained, each hidden unit becomes selectively activated by a specific data feature. Multiple combinations of features, with varying degrees of activation of the corresponding hidden units, allow for efficient generation of a large variety of new data samples. Tubiana and Monasson showed that ReLU hidden units allow RBMs to enter a compositional representation regime, in which each hidden unit encodes a limited set of features and the representation of sample is defined by a small set of hidden units with strong activations [39]. The existence of such encoding seems to depend on the values of the RBM parameters, such as the size of the hidden layer [13] and the weights sparsity [39]. However, Tubiana and Monasson use a slightly different variant of ReLU activation, which is associated with the potential

$$\mathcal{U}^{\mathrm{ReLU}}(h_i) = \begin{cases} \frac{h_i^2}{2} + c_i h_i & \text{if } h_i \geq 0 \\ +\infty & \text{if } h_i < 0, \end{cases} \tag{5.5}$$

so that the bias of the hidden units act as a threshold. Therefore, starting from the heuristic proposed by Hinton and Nair, we now approximate the states as $\max(0, \sum_j w_{ij} v_j - c_i + \mathcal{N}(0, \sigma(I_i)))$.

6 Experiments

6.1 Setup

6.1.1 Datasets and setup

We consider two benchmark problems. The *Bars & Stripes* (B&S) [24] data set consists of patterns of size $D \times D$ generated according to the following procedure. First, the pixels of each row are either set to zero or to one with equal probability. Then, with probability $1/2$ the pattern is rotated by 90 degrees. This leads to $N = 2^{D+1}$ patterns where the completely uniform patterns occur twice as often as the others. In particular, we will consider the 3×3 and the 4×4 patterns, see Figure 6.1.

The *MNIST* dataset [21] is a database of handwritten digits which is composed of a training set of 60,000 examples and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size grey-scale image of 28×28 pixels. The pixels take values in $[0, 255]$, but usually they are normalized to lie in $[0, 1]$. Then, we can directly use these normalized values as input for our model. Alternatively, we can binarize the values using a threshold of 0.5, so that all values below it are set to 0 and all the values above it are set to 1. Another alternative is to treat pixel values in $[0, 1]$ as probabilities of a binary event and use binomial units [17]. In this thesis we binarize the pixels values. Moreover, we will also consider the dataset which is obtained by flipping all the binary values of the MNIST, which is referred to as 1-MNIST. Some examples are shown in Figure 4.1.

If not stated otherwise, we use RBMs with 4 hidden units in the case of 3×3 B&S, 9 hidden units in the case of 4×4 B&S and 400 hidden units in the case of MNIST. These models are trained

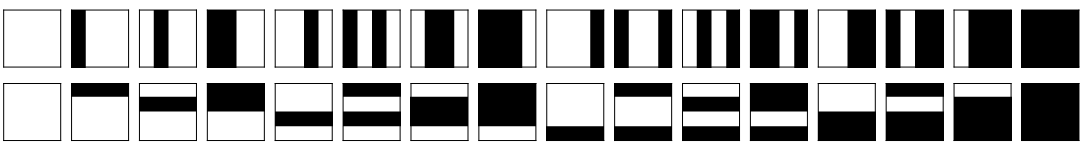
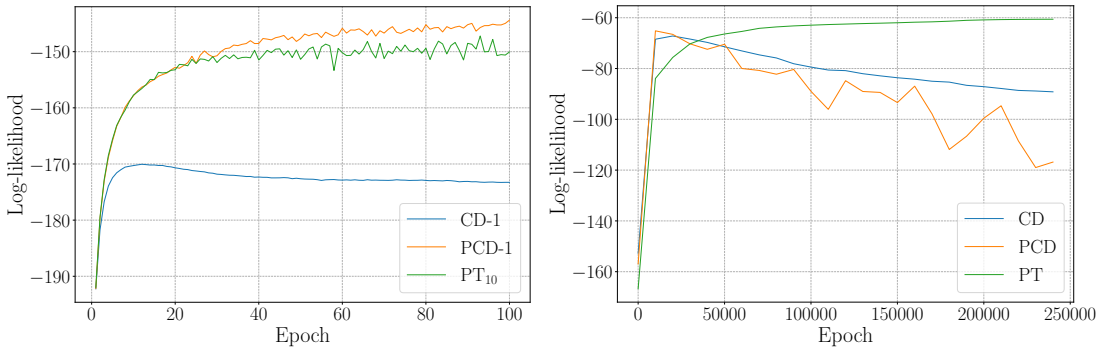


Figure 6.1: Patterns from the 4×4 B&S dataset.



(a) MNIST. (b) B&S.

Figure 6.2: Evolution of the average log-likelihood over 10 runs for CD, PCD and PT trainers on MNIST and 3×3 B&S datasets. In the case of B&S 4 hidden units are used, whereas for MNIST 16 hidden units are used.

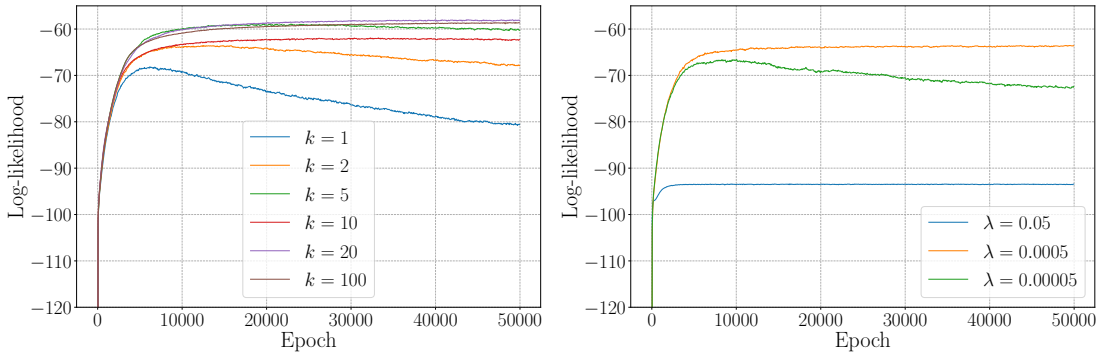
with standard batch learning for B&S and with mini-batches of 100 examples for MNIST. The weights are initialized using Xavier initialization [15], $w_{ij} = U[-\sqrt{6/(n+m)}, +\sqrt{6/(n+m)}]$, the initial hidden bias are set to zero and the initial visible bias are set to $\ln\big(\langle v_j \rangle_d / (1 - \langle v_j \rangle_d)\big)$, following Hinton recipe (see Section 3.7). The learning rate is set to $\eta = 0.1$ for B&S and to $\eta = 0.01$ for MNIST. To obtain unbiased comparisons between normal and centered RBMs no annealing learning rate, momentum or weight decay are used ($\nu = \lambda = 0$). Moreover, for a lighter notation, every time we write CD and PCD we mean CD-1 and PCD-1, and with PT we mean PT_{10}. The number of PCD chains is set by default to the same value of the mini-batch size.

6.2 Normal binary-binary RBMs

6.2.1 Comparison between CD, PCD and PT

First, we consider normal (i.e. non-centered) RBMs with binary units for both layers and we test the different trainers discussed in Section 3.6. We use RBMs with few units, so that the likelihood is tractable and we can monitor it. In particular, we use 4 hidden units for the 3×3 B&S and 16 hidden units for MNIST.

Figure 6.2 shows the comparison between the three types of trainer. The results are obtained by averaging over 10 trials. As we can see, CD shows divergence both for B&S and MNIST. That is, after an initial phase in which the log-likelihood increases, the likelihood starts to decrease. For the B&S, this happens also when using PCD. Such behaviour is due to the fact that during learning the weights tend to increase in magnitude. In fact, recall that the mixing rate of the Gibbs chain decreases as the magnitude of the RBM parameters increases, and hence the CD approximation gets more biased. Following Theorem 4, we can reduce the bias by increasing the number of Gibbs steps k. Indeed, the results obtained for different values of k clearly show that as k increases the divergence effect vanishes (Figure 6.3a). [ht] A better approach would be to gradually increase the number of Gibbs sampling steps as training proceeds. For example,



(a) Effect of CD-k for different values of k.

(b) Effect of weight decay

Figure 6.3: Average log-likelihood evolution over 10 runs of training on the B&S (a) with CD-k for different values of k and (b) with CD-1 for different values of the weight decay parameter λ.

one could start training with $k = 1$ and then gradually increase it to $k = 3, 10, \ldots$ as the weights grow. During this procedure it may be necessary also to reduce the learning rate, as the difference between the pairwise statistics that is used for learning will increase. However, as far as we know there are no fixed criteria that tell us when to change k, hence the only way is to proceed by trial and error.

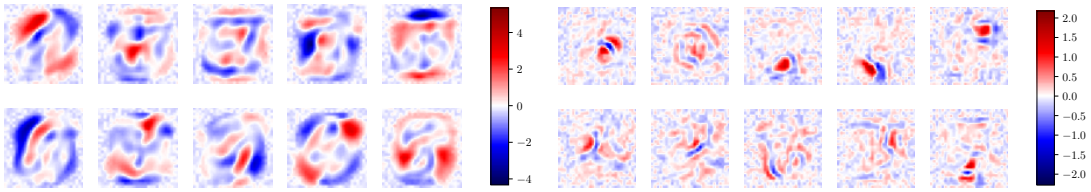
The divergence problem can be solved also by using weight decay with a proper value for the parameter λ. The results of training with different values of λ are shown in Figure 6.3b. As we can see, for $\lambda = 5{\cdot}10^{-2}$ and $\lambda = 5{\cdot}10^{-4}$ there is no divergence, whereas for $\lambda = 5{\cdot}10^{-5}$ divergence is present. However, for $\lambda = 5 \cdot 10^{-2}$ the log-likelihood is much lower than for $\lambda = 5 \cdot 10^{-4}$. This indicates that the choice of λ is critical not only for avoiding divergence but also for obtaining better performances in terms of log-likelihood.

Figure 6.2a also shows that, as expected, PCD and PT outperform CD. More surprisingly, PCD also performs slightly better than PT for MNIST. Probably this happens because of the learning parameters. Moreover, Figure 6.2b shows that PT does not suffer from divergence, thanks to the better mixing of the Gibbs chain.

6.2.2 Receptive fields

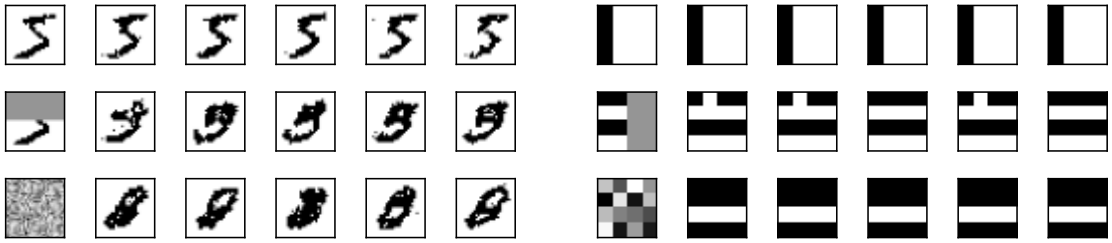
Hidden units can be seen as feature detectors. This is especially clear if we consider visible units that represent the pixels of an image. If we plot the color-coded weights of a hidden unit with the same shape as the input we can visualize the features that most activate that hidden unit. By doing so we visualize those that, in biological terms, would be the receptive fields of the neurons.

Figure 6.4 shows the receptive fields of RBMs with 16 and 400 hidden units. For the RBM with 16 hidden units, the fields appear quite articulate and sensible to most of the visible layer (Figure 6.4a). Conversely, the receptive fields of the RBM with 400 hidden units are more localized (Figure 6.4b): each single field is strongly activated by a circular spot. Thus, the RBM is able to construct more complex filters by joining units at higher and higher levels. For



(a) RBM with 16 hidden units. (b) RBM with 400 hidden units.

Figure 6.4: Examples of receptive fields of binary-binary RBMs with 16 and 400 hidden units trained on MNIST using PCD.



(a) MNIST. (b) 4×4 B&S.

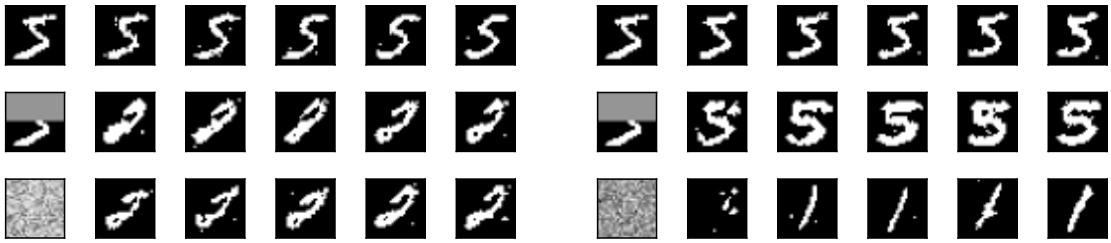
Figure 6.5: Examples of reconstructions of a test example, an incomplete example and generation starting from Gaussian noise (binary-binary RBM). For each row, the first column shows the initial input and the others show the samples from the Gibbs chain after 10 steps from the previous one, except for the third row, in which samples are spaced 1000 steps. The RBMs are trained using PCD.

example, for the generation of a handwritten digit image, it probably joins different units to obtain filters that respond to the oriented edges that compose the digit. Different combinations of features can produce different variants of the same digits. Many of those variants are not even contained in the training set, showing the generative power of RBMs.

6.2.3 Generative performances

RBMs are generative learning models. Therefore, once the RBM has been trained we can use it to generate new data starting from a given input. The input can be completely random, partially determined, or completely determined. In the second case, the RBM will hopefully perform a reconstruction of the input.

First, we start by giving as input some of the test examples. Then, we use an incomplete image as input and finally we feed the RBM with random Gaussian noise (mean $\mu = 0.5$, standard deviation $\sigma = 0.1$). Some examples of results are shown in Figure 6.5. For the MNIST dataset, the RBM is able to reproduce the given input but the reconstruction performances for the incomplete example and the random noise are quite weak. To some extent, the final outputs resemble a digit, but they are quite noisy and in some cases ambiguous or unclear. For the B&S dataset, which patterns are less complex than the MNIST ones, the RBM performs quite well in all tasks.



(a) Normal RBM. (b) Centered RBM (*dd*).

Figure 6.6: Examples of reconstructions of a test example, an incomplete example and generation starting from Gaussian noise for (a) normal binary-binary RBM; (b) *dd*-centered binary-binary RBM. For each row, the first column shows the initial input and the others show the samples from the Gibbs chain after 10 steps from the previous one, except for the third row, in which samples are spaced 1000 steps. The RBMs are trained using PCD.

6.3 Centered binary-binary RBMs

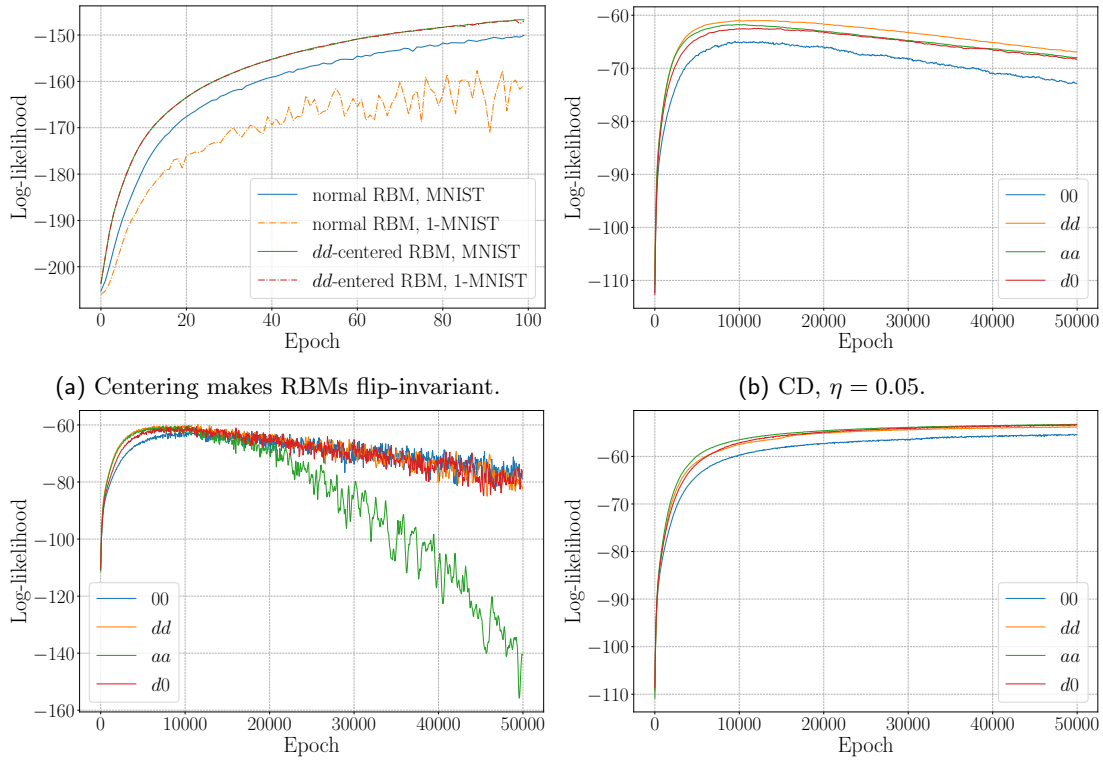
The types of centered RBMs we consider are listed in Table 6.1. The *original centered RBM* (*dd*) corresponds to the one introduced by Montavon and Müller [27], the *enhanced gradient RBM* is the first centered RBM, introduced by Cho et al. [6], and the *data normalization RBM* is the one introduced by Tang and Sutskever [34].

Acronym	μ	λ	Description
00	$\mathbf{0}$	$\mathbf{0}$	Normal RBM
dd	$\langle \boldsymbol{v} \rangle_d$	$\langle \boldsymbol{h} \rangle_d$	Original centered RBM
aa	$1/2(\langle \boldsymbol{v} \rangle_d + \langle \boldsymbol{v} \rangle_m)$	$1/2(\langle \boldsymbol{h} \rangle_d + \langle \boldsymbol{h} \rangle_m)$	Enhanced gradient RBM
*d*0	$\langle \boldsymbol{v} \rangle_d$	$\mathbf{0}$	Data normalization RBM

Table 6.1: Types of centered RBM.

First, we show that the centering trick makes RBMs flip-invariant. To this aim, we train two RBMs with 400 hidden units – one centered and the other not – using PCD on MNIST. We use a *dd*-centered RBM with a sliding factor of the moving average $\zeta = 0.01$. The evolution of the log-likelihood is shown in Figure 6.7a. As we can see, the log-likelihood is much lower in the case of the normal RBM trained on the 1-MNIST and it is also less stable. This asymmetry is confirmed also by Figure 6.6, which clearly shows that *dd*-centered RBM generates much better 1-MNIST samples. However, if we consider a centered RBM the performances on the two dataset become perfectly equivalent and moreover the likelihood is higher than the one obtained with the normal RBM. Thus, centering not only makes the RBM flip-invariant, but it also leads to a higher log-likelihood. Even if we show these facts using a *dd*-centered RBM, this holds for every type of centering listed in Table 6.1 [26].

The question that naturally arises is if there is a centering type that is better than the others. To find the answer to this question, we train all the different types of centered RBMs on the B&S dataset using all CD, PCD and PT. First of all, Figures 6.7b, 6.7c, 6.7d again show that



(a) Centering makes RBMs flip-invariant.

(b) CD, $\eta = 0.05$.

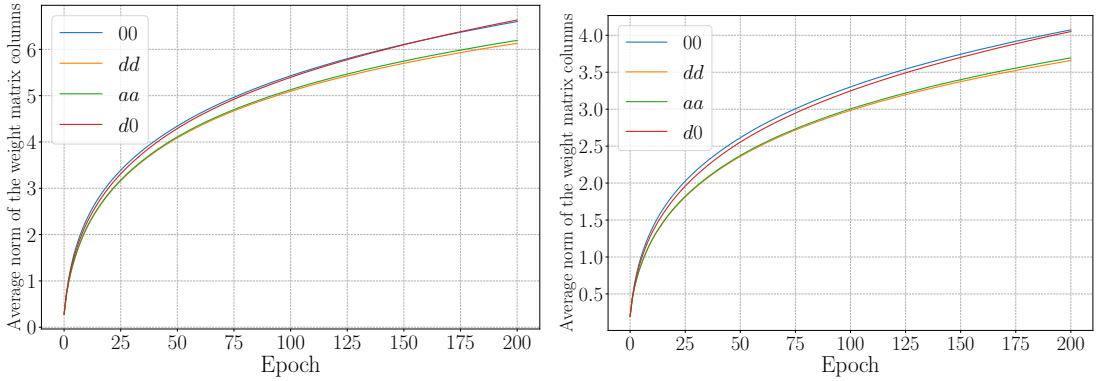
(c) PCD, $\eta = 0.05$.

(d) PT, $\eta = 0.01$.

Figure 6.7: (a) Evolution of the log-likelihood for normal and *dd*-centered RBMs with 400 hidden units trained on MNIST and 1-MNIST using PCD. (b,c,d) Evolution of the average log-likelihood on the 3×3 B&S dataset for different trainer types (η is the learning rate). The averages are computed over 10 runs.

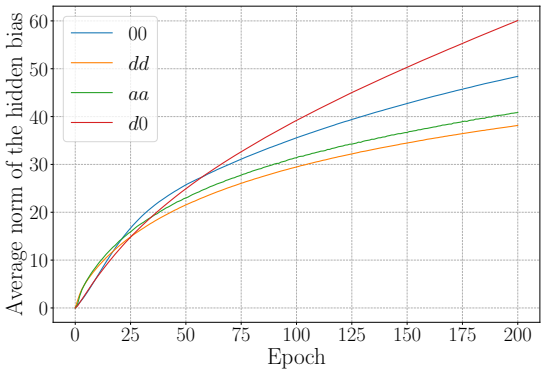
centering leads to better performances – that is, faster learning and higher log-likelihood – than normal RBMs. Furthermore, centering both the visible and the hidden variables (*dd*, *aa*) compared to centering only the visible variables (*d*0) accelerates learning and leads to a higher log-likelihood. Then, note that RBMs trained using CD and PCD show divergence, which can be more or less severe. Again, this is due to the bias induced by using just one step of Gibbs sampling. Therefore, in this case PCD seems not to increase the mixing rate sufficiently to avoid divergence. In fact, divergence is prevented when using PT, which leads to a faster mixing of the Gibbs chain.

Finally, we study the effect of centering on the RBM parameters. For this purpose, we compute the average weights and bias norms during training of RBMs with 400 hidden units on MNIST using PCD. The results are shown in Figure 6.8. Figure 6.8a and Figure 6.8b clearly illustrate that the row and column average norms of the weight matrix for *aa* and *dd* are smaller than for 00 and *d*0, meaning that it is not sufficient to center only the visible units. Figure 6.8c and Figure 6.8d show that the hidden bias for 00 and *d*0 are bigger than for *dd* and *aa*, whereas the visible bias are smaller and do not change significantly. This kind of behavior suggests that for 00 and *d*0 the bias values do not evolve properly during training, leaving the weights in charge of modeling the information.

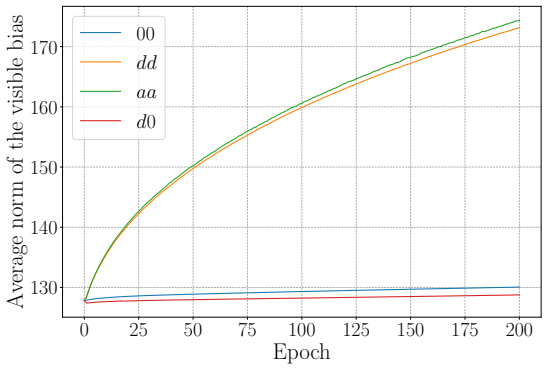


(a) Norm of the weight matrix columns.

(b) Norm of the weight matrix rows.



(c) Norm of the hidden bias.



(d) Norm of the visible bias.

Figure 6.8: Evolution of the average norm of normal and centered RBM parameters with 400 hidden units during training on MNIST using PCD. The averages are computed over 10 runs.

6.4 Binary-ReLU RBMs

Our binary-ReLU RBM is initialized following the paper by Tubiana and Monasson [39]. The weights are randomly initialized at $\pm\sqrt{0.1/m}$, the initial hidden bias are set to zero and the initial visible biases are initialized to $\ln(\langle v_j \rangle / (1 - v_j))$. Training is performed using PCD with 20 mini-batch size, 100 persistent chains, $k = 1$ Gibbs step between each update, 200 epochs and initial learning rate $\eta = 5 \cdot 10^{-3}$, which decays geometrically after 60 epochs to $\eta = 5 \cdot 10^{-4}$.

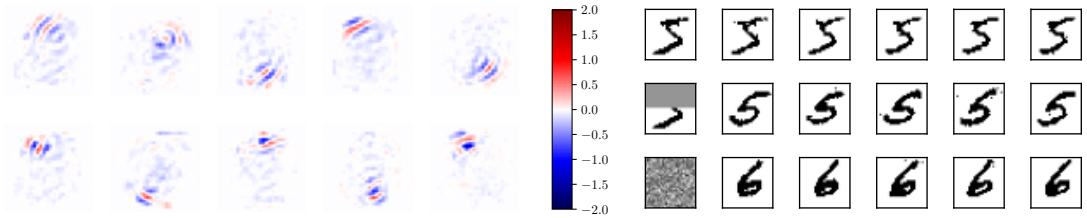
Learning is tracked by monitoring two parameters. The first one, referred to as *sparsity*, is a proxy for the fraction of non-zero weights:

$$\hat{p} = \frac{1}{mn} \sum_{i=1}^{n} \left[\frac{(\sum_{j=1}^{m} w_{ij}^2)^2}{\sum_{j=1}^{m} w_{ij}^4} \right], \tag{6.1}$$

whereas the second one is a proxy for the effective inverse temperature of the RBM distribution:

$$W_2 = \frac{1}{m} \sum_{i,j} w_{ij}^2, \tag{6.2}$$

so that an estimator of the temperature T in $p(\boldsymbol{v}, \boldsymbol{h}) = e^{-\frac{E(\boldsymbol{v},\boldsymbol{h})}{T}}$ is given by $\hat{T} = \hat{p}/W_2$. To



(a) Receptive fields. (b) Reconstructions.

Figure 6.9: Examples of (a) receptive fields and (b) reconstructions starting from a test example, an incomplete example and Gaussian noise of a binary-ReLU RBM. The receptive field structure clearly illustrates the so called compositional phase, in which only a small subset of hidden units are active. For each row in (b), the first column shows the initial input and the others show the samples from the Gibbs chain after 10 steps from the previous one, except for the third row, in which samples are spaced 1000 steps.

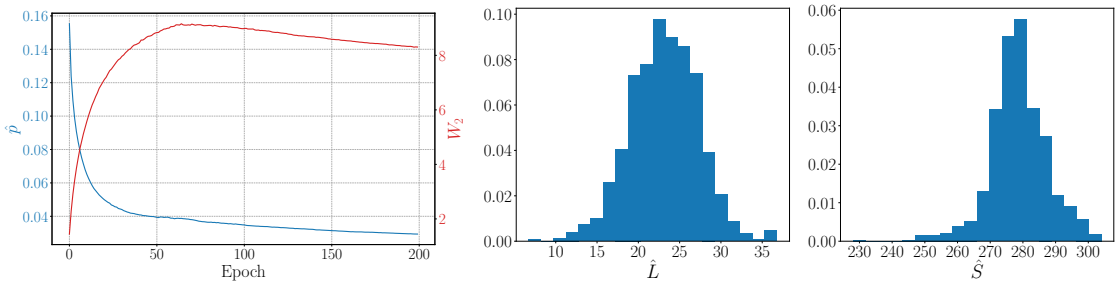
control the weight sparsity $\hat{p}$, Tubiana and Monasson consider a regularization penalty given by

$$L^{(x)} = \frac{\lambda_x}{x} \sum_{i=1}^{n} \left[\sum_{j=1}^{m} |w_{ij}| \right]^x \tag{6.3}$$

The case $x = 1$ is the usual L_1 penalty, which is known to sparse weight matrices. However, this kind of penalty may lead to hidden units that are completely disconnected from the visible layer, making them useless. To avoid this, it is better to choose $x = 2$ or $x = 3$. This is equivalent to a "custom" L_1 penalty in which hidden units that are strongly coupled with the visible ones get a stronger regularization, and vice versa. This should promote homogeneity among hidden units. In particular, we choose $x = 2$.

Figure 6.9a shows the receptive fields after learning. Each feature looks like an elementary stroke which is extremely localized around a small portion of the visible layer. Notice the difference with Figure 6.4b, in which the features are localized as well, but much less than in this case. These strokes are then combined by the RBM to obtain more complex patterns, i.e. the digits. Moreover, Figure 6.9a clearly illustrates the fact that the RBM operates in what Tubiana and Monasson call the compositional phase: most hidden units are silent and only few ones are strongly activated. Indeed, this particular regime is characterized by two quantities: the number of silent units $h_i = 0$, denoted by $\hat{S}$, and the participation ratio $\hat{L} = [(\sum_i h_i^3)^2 / \sum_i h_i^6]$, which gives an estimate of the number of strongly activated units. Looking at Figure 6.10b and Figure 6.10c, we see that, on average, each generated handwritten digit image is composed by $\hat{L} \approx 22$ elementary strokes whereas $\hat{S} \approx 280$ hidden units are silent. The presence of a compositional phase is also confirmed by Figure 6.10a, which shows that during training the sparsity parameter $\hat{p}$ decreases, whereas W_2 increases. Sparsity is good because it prevents overfitting and allows better interpretation of the weights.

Nair and Hinton first found that rectified linear units improve RBMs [28]. This happens because, unlike binary units, rectified linear units preserve information about the magnitudes of their inputs above threshold. A comparison between Figure 6.9b and Figure 6.5a clearly confirms



(a) Sparsity $\hat{p}$ and effective inverse temperature W_2.

(b) PDF of participation ratio $\hat{L}$.

(c) PDF of the number of silent hidden units $\hat{S}$.

Figure 6.10: (a) Plots of learning parameters $\hat{p}$ (sparsity) and W_2 (effective inverse temperature) and PDFs of (b) participation ratio $\hat{L}$ (c) number of silent hidden units $\hat{S}$ for binary-ReLU RBM. The histograms are obtained by measuring $\hat{L}$ and $\hat{S}$ for the digits generated starting from 1000 samples of Gaussian noise.

this, showing that binary-ReLU RBMs have better generative performances then binary-binary RBMs in all tasks.

7 Conclusions

In this thesis we presented the theoretical framework behind RBMs, that is Probabilistic Graphical Models and Markov Random Fields. Then we discussed different types of RBMs and training algorithms. First, we studied binary-valued RBMs, finding that in some cases CD and PCD display divergence, leading to a decrease of the likelihood. In these cases, PT can solve the problem, but it introduces computational overhead. Other solutions could be to gradually increase the number of Gibbs sampling steps or to use weight decay. We also tested the generative performances of such models, using RBMs to recreate, reconstruct and generate new examples.

Then we considered centered binary RBMs, showing that – unlike normal binary RBMs – they are able to reach the same performances on MNIST and 1-MNIST datasets thanks to their flip-invariance. We also showed that centering not only makes RBMs flip-invariant, but it also leads to higher likelihood. Moreover, through other tests we showed that RBMs in which both visible and hidden variables are centered form more accurate models of the data distribution than normal RBMs and RBMs in which only the visible variables are centered. In particular, it seems that the best centering type is *dd*, which computes the offsets as the mean of the variables over the data.

Afterwards, we introduced binary-ReLU RBMs, showing that by using rectified linear hidden units and a regularization which promotes homogeneity the RBM operates in the so-called compositional phase. In this regime, when a sample is generated most hidden units are silent and only few ones are strongly activated. These kinds of models allow a good degree of interpretability and also show better generative performances.

This work is also intended to show how simple models like RBMs can have huge representational power while maintaining a good degree of interpretability. An extraordinary example of this fact is the article by Tubiana, Cocco and Monasson [38], in which they use an RBM with ReLU potential (more precisely a double-ReLU potential, which consists of the combination of two ReLUs) to model protein families from sequence data. Moreover, the features inferred by the RBM are biologically interpretable: they are related to structure, to function or to phylogenetic identity. In addition, since RBMs are generative models, they are able to design new protein

sequences.

RBMs are also the building blocks of Deep Boltzmann Machines (DBMs), deep undirected graphical models with several hidden layers where successive layers have a bipartite connectivity structure. Therefore, studying RBMs can be useful to design DBMs with better performances. In fact, a centering optimization method was proposed by Montavon et al. [27] to make the learning mechanism more stable and also for the purpose of designing generative, faster and discriminative models.

Bibliography

[1] Adriano Barra et al. “On the equivalence of hopfield networks and boltzmann machines”. In: *Neural Networks* 34 (2012), pp. 1–9 (cit. on p. 34).

[2] Y. Bengio and Olivier Delalleau. “Justifying and Generalizing Contrastive Divergence”. In: *Neural computation* 21 (Dec. 2008), pp. 1601–21. DOI: 10.1162/neco.2008.11-07-647 (cit. on p. 20).

[3] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738 (cit. on pp. 3, 4, 6, 10).

[4] Pierre Brémaud. *Markov chains: Gibbs fields, Monte Carlo simulation, and queues*. Vol. 31. Springer Science & Business Media, 2013 (cit. on pp. 10, 11).

[5] KyungHyun Cho, Alexander Ilin, and Tapani Raiko. “Improved Learning of Gaussian-Bernoulli Restricted Boltzmann Machines”. In: June 2011, pp. 10–17. ISBN: 978-3-642-21734-0. DOI: 10.1007/978-3-642-21735-7_2 (cit. on p. 34).

[6] KyungHyun Cho, Tapani Raiko, and Alexander Ilin. “Enhanced gradient and adaptive learning rate for training restricted Boltzmann machines”. In: *ICML*. 2011 (cit. on pp. 27, 41).

[7] Peter Clifford. “Markov random fields in statistics”. In: *Disorder in physical systems: A volume in honour of John M. Hammersley* (1990), pp. 19–32 (cit. on p. 5).

[8] Francis Crick and Graeme Mitchison. “The function of dream sleep”. In: *Nature* 304.5922 (1983), pp. 111–114 (cit. on p. 9).

[9] Geoffrey E. Hinton. “Training Products of Experts by Minimizing Contrastive Divergence”. In: *Neural Computation (NECO)* 14.8 (2002), pp. 1771–1800 (cit. on pp. 19, 20).

[10] Guillaume Desjardins et al. “Parallel Tempering for Training of Restricted Boltzmann Machines”. In: 2010 (cit. on p. 23).

[11] Asia Fischer and Christian Igel. “Bounding the Bias of Contrastive Divergence Learning”. In: *Neural Comput.* 23.3 (Mar. 2011), pp. 664–673. ISSN: 0899-7667. DOI: 10.1162/NECO_a_00085. URL: https://doi.org/10.1162/NECO_a_00085 (cit. on p. 20).

[12] Asja Fischer and Christian Igel. "Empirical Analysis of the Divergence of Gibbs Sampling Based Learning Algorithms for Restricted Boltzmann Machines". In: Sept. 2010, pp. 208–217. ISBN: 978-3-642-15824-7. DOI: 10.1007/978-3-642-15825-4_26 (cit. on pp. 4, 21, 22).

[13] Asja Fischer and Christian Igel. "Training restricted Boltzmann machines: An introduction". In: *Pattern Recognition* 47.1 (2014), pp. 25–39 (cit. on pp. 4, 5, 7–9, 11, 15, 17, 20, 21, 23, 27, 36).

[14] Stuart Geman and Donald Geman. "Stochastic relaxation, Gibbs distributions, and the Bayesian restoration of images". In: *IEEE Transactions on pattern analysis and machine intelligence* 6 (1984), pp. 721–741 (cit. on p. 11).

[15] Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks". In: *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Yee Whye Teh and Mike Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256. URL: http://proceedings.mlr.press/v9/glorot10a.html (cit. on pp. 24, 38).

[16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016 (cit. on pp. 8, 9, 17).

[17] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. "A fast learning algorithm for deep belief nets". In: *Neural computation* 18.7 (2006), pp. 1527–1554 (cit. on p. 37).

[18] Geoffrey E. Hinton. "A Practical Guide to Training Restricted Boltzmann Machines". In: *Neural Networks: Tricks of the Trade: Second Edition*. Ed. by Grégoire Montavon, Geneviève B. Orr, and Klaus-Robert Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 599–619. ISBN: 978-3-642-35289-8. DOI: 10.1007/978-3-642-35289-8_32. URL: https://doi.org/10.1007/978-3-642-35289-8_32 (cit. on pp. 17, 24, 35).

[19] Daphne Koller and Nir Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009 (cit. on pp. 3, 4, 10, 11).

[20] Solomon Kullback and Richard A Leibler. "On information and sufficiency". In: *The annals of mathematical statistics* 22.1 (1951), pp. 79–86 (cit. on p. 7).

[21] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 37).

[22] Yann A LeCun et al. "Efficient backprop". In: *Neural networks: Tricks of the trade*. Springer, 2012, pp. 9–48 (cit. on p. 27).

[23] Philip Long and Rocco Servedio. "Restricted Boltzmann Machines are Hard to Approximately Evaluate or Simulate". In: Aug. 2010, pp. 703–710 (cit. on p. 14).

[24] David JC MacKay and David JC Mac Kay. *Information theory, inference and learning algorithms*. Cambridge university press, 2003 (cit. on p. 37).

[25] Pankaj Mehta et al. "A high-bias, low-variance introduction to Machine Learning for physicists". In: *Physics Reports* 810 (May 2019), pp. 1–124. ISSN: 0370-1573. DOI: 10.1016/j.physrep.2019.03.001. URL: http://dx.doi.org/10.1016/j.physrep.2019.03.001 (cit. on pp. 8, 14).

[26] Jan Melchior, Asja Fischer, and Laurenz Wiskott. "How to center deep Boltzmann machines". In: *The Journal of Machine Learning Research* 17.1 (2016), pp. 3387–3447 (cit. on pp. 1, 27, 29–31, 41).

[27] Grégoire Montavon and Klaus-Robert Müller. "Deep Boltzmann machines and the centering trick". In: *Neural networks: tricks of the trade*. Springer, 2012, pp. 621–637 (cit. on pp. 27, 29, 41, 48).

[28] Vinod Nair and Geoffrey E Hinton. "Rectified linear units improve restricted boltzmann machines". In: *Icml*. 2010 (cit. on pp. 1, 35, 44).

[29] Siamak Ravanbakhsh et al. "Stochastic neural networks with monotonic activation functions". In: *Artificial Intelligence and Statistics*. PMLR. 2016, pp. 809–818 (cit. on p. 35).

[30] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. "Learning representations by back-propagating errors". In: *nature* 323.6088 (1986), pp. 533–536 (cit. on p. 8).

[31] Ruslan Salakhutdinov. "Learning in Markov Random Fields using Tempered Transitions." In: vol. 22. Jan. 2009, pp. 1598–1606 (cit. on p. 23).

[32] Nicol N Schraudolph. "Centering neural network gradient factors". In: *Neural Networks: Tricks of the Trade*. Springer, 1998, pp. 207–226 (cit. on p. 27).

[33] Paul Smolensky. *Information processing in dynamical systems: Foundations of harmony theory*. Tech. rep. Colorado Univ at Boulder Dept of Computer Science, 1986 (cit. on p. 13).

[34] Yichuan Tang and Ilya Sutskever. "Data normalization in the learning of restricted Boltzmann machines". In: *Department of Computer Science, University of Toronto, Technical Report UTML-TR-11-2* (2011) (cit. on pp. 27, 41).

[35] Yee Whye Teh and Geoffrey E Hinton. "Rate-coded restricted Boltzmann machines for face recognition". In: *Advances in neural information processing systems* (2001), pp. 908–914 (cit. on p. 34).

[36] Tijmen Tieleman. "Training Restricted Boltzmann Machines Using Approximations to the Likelihood Gradient". In: *Proceedings of the 25th International Conference on Machine Learning*. ICML '08. Helsinki, Finland: Association for Computing Machinery, 2008, pp. 1064–1071. ISBN: 9781605582054. DOI: 10.1145/1390156.1390290. URL: https://doi.org/10.1145/1390156.1390290 (cit. on p. 22).

[37] Alessandro Treves. "Quantitative estimate of the information relayed by the Schaffer collaterals". In: *Journal of Computational Neuroscience* 2.3 (1995), pp. 259–272. DOI: 10.1007/BF00961437. URL: https://doi.org/10.1007/BF00961437 (cit. on p. 35).

[38] Jérôme Tubiana, Simona Cocco, and Rémi Monasson. "Learning protein constitutive motifs from sequence data". In: *Elife* 8 (2019), e39397 (cit. on p. 47).

[39] Jérôme Tubiana and Rémi Monasson. "Emergence of compositional representations in restricted Boltzmann machines". In: *Physical review letters* 118.13 (2017), p. 138301 (cit. on pp. 36, 43).