Politecnico di Torino

Master's Degree in

Physics of Complex Systems

Master's Degree Thesis

Training Gaussian Restricted Boltzmann machines using Expectation Propagation

Supervisor:

Anna Paola Muntoni

Candidate:

Roberto Puntorieri

July 2024

Abstract

This thesis investigates a new training technique for Restricted Boltzmann Machines (RBMs), a type of stochastic neural network employed in unsupervised learning. Specifically, the focus of this work is on RBMs with a Gaussian prior distribution for the hidden units, leading to a training technique that depends exclusively on the visible units provided as input. The method relies on Expectation Propagation (EP), a Bayesian inference technique designed to approximate intractable distributions. As a testing ground, we analyze its performance on the MNIST dataset, a large database of handwritten digits commonly used for training and testing machine learning algorithms.

First, we begin by detailing the historical background and the foundational concepts of RBMs, followed by a similar exposition for EP. Secondly, we present the mathematical steps employed for implementing the EP formalism to RBMs. Afterward, we train our model on the digits 0 and 1 of the MNIST dataset, leveraging many independent copies of the approximation to improve the efficacy of the algorithm. After identifying the optimal conditions for efficient training and achieving satisfactory results, we employ the exploration of the dataset space achieved at convergence to cluster the dataset. A similar analysis is then performed for the more complex case of digits ranging from 0 to 4.

Finally, we investigate the potential of integrating a population dynamics scheme into RBM training, allowing multiple independent copies of the system to evolve and interact with each other, exchanging information to explore the solution space more effectively.

Acknowledgments

I am deeply grateful to my supervisor Anna Paola Muntoni for her constant support and guidance throughout this thesis. Her kindness and patience have been incredibly valuable to me, as she always took the time to listen and provide helpful feedback. I would also like to thank all the PhD students, Francesco, Leonardo, and Mattia, for their hospitality and companionship during these months. A special thanks goes to Francesco for his support and assistance, not only during the thesis period but throughout my entire Master's program.

Contents

List of Figures

Chapter 1

An Introduction to Restricted Boltzmann Machines

1.1 Historical Background

The roots of Restricted Boltzmann machines (RBMs) can be traced back to systems consisting of interacting binary variables, initially proposed as toy models of condensed matter systems in statistical physics.

One of the most remarkable examples is the Ising model, originally introduced by Wilhelm Lenz as a model for ferromagnetism, and first solved in its one-dimensional case by Ernst Ising in his PhD Thesis [1].

The Ising model consists of binary variables, also called "spins", that can be in two possible states (+1 or -1). Spins are arranged in a lattice and can interact with their neighboring spins. This interaction between two adjacent sites i, j is represented by the matrix element J_{ij} and each spin can also be coupled with an external magnetic field h_i. The energy of a configuration of N spins $\boldsymbol{\sigma} = \{\sigma_1, ..., \sigma_N\}$ is given by the renowned Hamiltonian function:

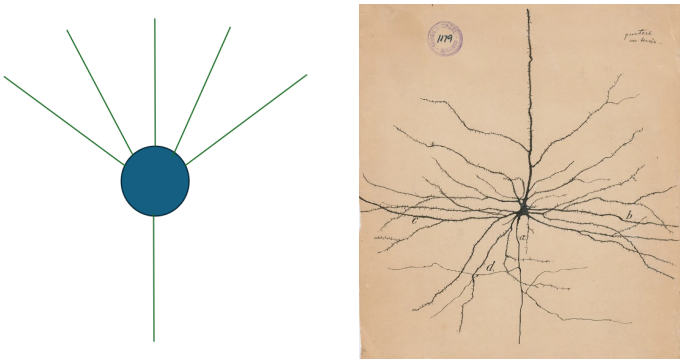
$$H(\boldsymbol{\sigma}) = -\sum_{<ij>} J_{ij}\sigma_i\sigma_j - \sum_i h_i\sigma_i$$

where $< ij >$ denotes that the sites i and j are nearest neighbors.

Despite its simplicity, the Ising model does not only predict crucial aspects of ferromagnetism but it can also be successfully applied to a wide class of systems and problems (e.g. the lattice gas and the condensation transition, the helix–coil transition and other order-disorder phenomena in biological systems).

During the 1980s, these type of models consisting of interacting binary variables made their debut in the domains of neuroscience and artificial intelligence. In particular, the Hopfield model, first described by Shun'ichi Amari in 1972 [2] and popularised by John Hopfield in 1982 [3] , emerged as a prominent example within this context.

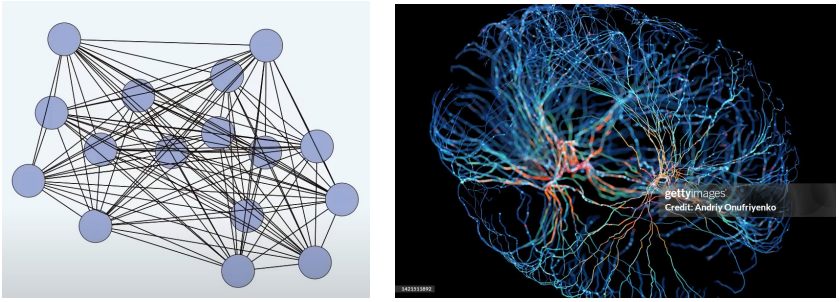
The Hopfield network was inspired by the functioning of the human brain and it consists of interconnected "neurons" (see Fig. 1.1), where each neuron is represented by a binary unit s_i that can be either "on" (+1) or "off" (-1), depending on whether the unit's input exceed a certain threshold U_i. The connection between two neurons i and j is represented by the "synaptic weight" T_{ij}, which can be positive or negative, indicating an excitatory or inhibitory connection, respectively.



(a) (b)

Figure 1.1: (a) Schematic representation of a Hopfield neuron and the synaptic weights through which neurons interact, resembling biological axon-synapse-dendrite connections. (b) Neuron illustration by Santiago Ramón y Cajal, The pyramidal neuron of the cerebral cortex, 1904 Ink and pencil on paper, 8 5/8 x 6 7/8 in. Credit: Cajal Institute (CSIC), Madrid

The Hopfield network exhibits similarities with the way the brain stores and retrieves information (see Fig. 1.2). Indeed in the brain memories are not stored in isolated locations but are distributed across a network of neurons, and similarly Hopfield networks are capable of storing patterns ('memories') of activation and retrieving them under noisy conditions.



(a) (b)

Figure 1.2: (a) Graphical example of a recurrent neural network modeling the functioning of the brain. Credits: How are memories stored in neural networks?|The Hopfield Network (b) Pictorial representation of the synaptic connections in the brain. Credits: [Andriy Onufriyenko]/[Moment] via Getty Images

Inspired by the Hopfield network, in 1983 Hinton et al. [4] proposed the Boltzmann Machine (BM), describing it as a network of coupled binary units able to learn the underlying constraints of a domain simply by being shown examples from the domain. The network adjusts its connections to create a model that can generate instances with the same (approximated) probability distribution of the examples shown. BMs proved capable to successfully complete pattern tasks on small-scale examples, but the learning algorithm was prohibitively slow. A special case of BM, called the Restricted Boltzmann machine[1] (RBM), brought BMs back into the spotlight thanks to a fast training algorithm (specific for RBMs) called the Contrastive Divergence algorithm, proposed by Hinton et al. in 2002 [6]. Later, Hinton et al. showed that combining RBMs on top of each other was an effective method for learning deep representations [7].
In the following section, we are going to analyze the key features and the mathematical formalism underlying Restricted Boltzmann Machines.

[1]RBMs were originally introduced within the framework of the theory of language and symbolic computation under the name Harmonium by Paul Smolensky [5]

1.2 Definitions

1.2.1 Boltzmann Machines and Restricted Boltzmann Machines

A Restricted Boltzmann machine (RBM) is a probabilistic graphical model [2] utilized to learn the core features of an unknown target distribution by means of the samples generated by the distribution under investigation.

As their name implies, RBMs are a special case of Boltzmann machines (BMs). Therefore, before exploring in detail RBMs, it is fundamental to understand the basic concepts behind the BM.

A BM is a parameterized generative model that represents a probability distribution. It consists solely of one type of units, whose interactions can be depicted by undirected weighted edges.

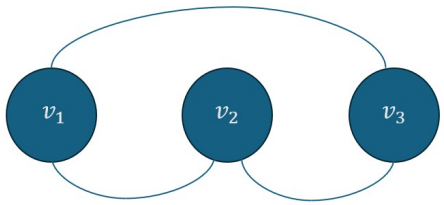


Figure 1.3: Graphical representation of a BM example.

Learning a BM means, given a training dataset composed of some observations, adjusting the BM parameters until the learned probability distribution fits the target distribution as well as possible. Regrettably, in practical machine learning problems, BMs learning algorithms become prohibitively slow due to the vast number of potential connections that emerge.

RBMs tackle this issue by imposing an additional constraint on the topology of the network: the units must form a bipartite graph so that two distinct lay-

[2] i.e., a probability distribution over a multidimensional space, defined via an interaction graph

ers can be distinguished, the visible and the hidden layer, composed of visible and hidden units, respectively. Consequently, according to the definition of a bipartite graph, interactions between nodes belonging to the same layer are prohibited. The visible units correspond to the components of an observation (e.g. the pixels of a digital image), whereas the hidden units capture the complex relationships among the input variables (e.g. in an image, the hidden units typically represent higher-level features or patterns emerging from the pixel-level representation).

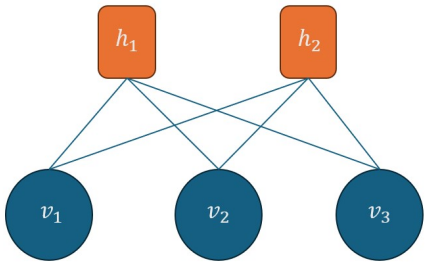


Figure 1.4: Graphical representation of an RBM example.

1.2.2 Bayesian Inference

Frequently, in our problems, the information we have to deal with is partial or affected by uncertainty and noise. Therefore, due to the intrinsically uncertain nature of this kind of problems, we need to investigate them within a probabilistic framework.

Statistical inference is the process that allows us to infer the properties of the distribution under investigation by analyzing the limited data at our disposal. In particular, in the context of RBMs, Bayesian inference provides a powerful framework for parameter estimation.

Bayesian inference is a method of statistical inference, probably the most intuitive, that utilizes Bayes' theorem to update our prior belief about unknown parameters in light of new evidences (the observed data).

Mathematically, Bayes' theorem can be expressed as follows:

$$P(H \mid E) = \frac{P(E \mid H) \cdot P(H)}{P(E)} \propto P(E \mid H) \cdot P(H)$$

where:

- $P(H)$ is referred to as the *prior probability*, where H denotes a specific hypothesis (i.e. a set of parameters) describing the model under investigation.
- $P(E)$ is usually called the *marginal likelihood* and it is the normalization constant. Here, E stands for "evidence" and denotes the new observed data that were not considered when calculating the prior probability.
- $P(H|E)$ is called the *posterior probability* and it gives us the updated probability of H, after observing E.
- $P(E|H)$ is called the *likelihood*, indicating how likely the data are under the assumption that the hypothesis is true.

By incorporating prior distributions over RBM parameters, Bayesian methods enable the integration of prior knowledge and the quantification of uncertainty. This approach contrasts with traditional maximum likelihood estimation by offering a full posterior distribution of parameters, allowing for more robust model training and improved generalization. Techniques such as variational inference further enhance computational efficiency, making Bayesian inference feasible for large-scale RBM applications.

1.3 Types of Units

In this section, we will explore the various types of units that can be used when training RBMs. Depending on the nature of the problem we are facing, choosing the appropriate unit type is very important since it has great influence on how the model interprets and generates data.

1.3.1 Binary Units

Binary units, which are suitable when working with binary data, are commonly modeled using the Bernoulli distribution. The activation function[3] for these units is the logistic sigmoid function $\sigma(x) \in [0, 1]$, such that:

$$P(x) = \sigma(x) = \frac{1}{1 + e^{-x}} = \frac{e^x}{e^x + e^0} \tag{1.1}$$

1.3.2 Softmax units

In we have N_U possible states, we can generalize the binary case using:

$$P_u(x_u) = \frac{e^{x_u}}{\sum_{u=1}^{N_U} e^{x_u}} \tag{1.2}$$

This is referred to as a "softmax" unit and it is useful when the data can assume many possible discrete values.

1.3.3 Continuous Data Modeling Units

To deal with real-valued data, binary units are not the most suitable option. In this case, one can opt for Gaussian units, which use a Gaussian distribution $\mathcal{N}(\mu, \sigma)$ as prior, where μ and σ represent the mean and the standard deviation, respectively. This type of unit represents continuous values effectively, and the activation function typically employed for these units is the linear one: $f(x) = x$. Another option for continuous data is binomial units, which are characterized by noisy integer values ranging from 0 to N. These units can be obtained, as shown in [8], by creating N identical copies of a binary unit, each sharing identical weights and biases. Additionally, if we maintain the same weights and biases for each copy but introduce a fixed offset to the biases, we obtain the so-called rectified linear units (ReLU). For example, in [9] Hinton and Nair showed

[3] Activation functions are mathematical functions applied to the input values of a unit to determine its output. These functions introduce non-linearity into the model, which allows the neural network to learn more complex patterns.

that employing ReLU units for training RBMs on the NORB (NYU[4] Object Recognition Benchmark) dataset, yielded improved feature learning compared to using binary units.

1.4 Binary Vector RBMs

RBMs were originally devised with binary visible and hidden units.
For instance, let us consider a training dataset comprising binary images. As previously mentioned, the RBM's visible units correspond to pixels, whereas the hidden neurons extract relevant features from the observations.
Given N visible units $\mathbf{v} = (v_1, ..., v_N)$ and M hidden units $\mathbf{h} = (h_1, ..., h_M)$, the energy of a joint configuration ($\mathbf{v}$,$\mathbf{h}$) is given by:

$$E(\mathbf{v}, \mathbf{h}) = -\sum_{i=1}^{N} a_i v_i - \sum_{j=1}^{M} b_j h_j - \sum_{i=1}^{N}\sum_{j=1}^{M} v_i h_j W_{ij} \tag{1.3}$$

where, a_i and b_j represent the biases of the i-th visible unit and the j-th hidden unit, respectively, while W_{ij} denotes the weight modeling their interaction.
This energy contributes to the joint probability distribution $P(\mathbf{v},\mathbf{h})$ which has the form of a Boltzmann (or Gibbs) distribution:

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

where

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

is the normalization factor, also referred to as the partition function in statistical physics.

By summing over all possible hidden layer configurations $\mathbf{h}$, one can retrieve the marginal distribution over the visible layer $\mathbf{v}$:

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

[4] "New York University"

Notice also that since for an RBM two variables of the same layer have no connections, the conditional distributions $P(\mathbf{h}|\mathbf{v})$ and $P(\mathbf{v}|\mathbf{h})$ factorize nicely:

$$P(\mathbf{h}|\mathbf{v}) = \prod_{i=1}^{N} P(h_i|\mathbf{v}) \ , \ P(\mathbf{h}|\mathbf{v}) = \prod_{i=1}^{N} P(h_i|\mathbf{v}) \tag{1.4}$$

1.5 Unsupervised Learning for Markov Random Fields

Both BMs and RBMs are probabilistic graphical models, specifically undirected graphical models also known as Markov random fields[5] (MRFs) [10]. Therefore, prior to discussing the specific learning algorithms for RBMs, it is essential to examine unsupervised learning within the more general context of MRFs. Unsupervised learning is a machine learning method in which algorithms learn (significant aspects of) an unknown distribution $P(\mathbf{x})$ using sample data. Given a graphical model whose structure is known and whose associated energy function is parameterized by $\boldsymbol{\theta}$, our objective is to tune these parameters to accurately model the desired but unknown distribution. Let us start by considering a training dataset $D = (\mathbf{v^{(1)}}, ..., \mathbf{v^{(N_D)}})$ composed of N_D independent and identically distributed (i.i.d.) data samples $\mathbf{v^{(i)}}$. Typically, to derive the parameters for our statistical model, one employs maximum-likelihood estimation. Specifically, in the context of MRFs, training corresponds to find the parameters $\boldsymbol{\theta}$ that maximize the likelihood:

$$\mathcal{L}(D|\boldsymbol{\theta}) = \prod_{i=1}^{N_D} P_i(\mathbf{v^{(i)}}|\boldsymbol{\theta})$$

Thanks to the monotonic behavior of the logarithmic function, maximizing the likelihood is equivalent to maximize the log-likelihood:

$$\ln \mathcal{L}(D|\boldsymbol{\theta}) = \ln \prod_{i=1}^{N_D} P_i(\mathbf{v^{(i)}}|\boldsymbol{\theta}) = \sum_{i=1}^{N_D} \ln P_i(\mathbf{v^{(i)}}|\boldsymbol{\theta})$$

[5]The key property of an MRF is the Markov property (w.r.t. the graph), which states that each random variable is conditionally independent of all the other variables in the graph, given its neighboring variables.

In turn, maximizing the log-likelihood corresponds to minimizing the Kullback-Leibler (KL) divergence[6] between the target probability distribution $P(\mathbf{x})$ and its approximating distribution.

In general, as shown for example in [11], the probability distribution of every MRF can be expressed in the form of a Gibbs distribution. However, obtaining the maximum likelihood parameters analytically is typically not feasible for such a distribution. Therefore, optimization methods such as gradient ascent on the log-likelihood need to be employed. This corresponds to iteratively update the parameters $\boldsymbol{\theta}$ as follows:

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} + \eta \frac{\partial}{\partial \boldsymbol{\theta}^{(t)}} (\ln \mathcal{L}(D|\boldsymbol{\theta}^{(t)}))$$

where η is the learning rate that determines the size of the steps taken in the direction of the gradient during each iteration.

Since RBMs are a type of MRFs with hidden variables, it is interesting to look at MRFs in which, for a given data sample d, the set of variables $\mathbf{x}^{(\mathbf{d})} = (\mathbf{v}^{(\mathbf{d})}, \mathbf{h})$ is divided into visible variables $\mathbf{v}^{(\mathbf{d})} = (v_1^{(d)}, ..., v_n^{(d)})$ corresponding to the components of the d-th observation, and hidden variables $\mathbf{h} = (h_1, ..., h_m)$, which are assumed to be discrete and capture the dependencies between visible variables. Therefore, focusing on a single data sample $\mathbf{v}^{(\mathbf{d})}$:

$$\ln \mathcal{L}(\mathbf{v}^{(\mathbf{d})}|\boldsymbol{\theta}) = \ln P(\mathbf{v}^{(\mathbf{d})}|\boldsymbol{\theta}) = \ln \frac{1}{Z} \sum_{\mathbf{h}} e^{-E(\mathbf{v}^{(\mathbf{d})}, \mathbf{h})} = \ln \sum_{\mathbf{h}} e^{-E(\mathbf{v}^{(\mathbf{d})}, \mathbf{h})} - \ln \sum_{\mathbf{v}^{(\mathbf{d})}, \mathbf{h}} e^{-E(\mathbf{v}^{(\mathbf{d})}, \mathbf{h})}$$

And if we exploit the fact that the conditional probability can be written as follows:

$$P(\mathbf{h}|\mathbf{v}^{(\mathbf{d})}) = \frac{P(\mathbf{v}^{(\mathbf{d})}, \mathbf{h})}{P(\mathbf{v}^{(\mathbf{d})})} = \frac{\cancel{\frac{1}{Z}} e^{-E(\mathbf{v}^{(\mathbf{d})}, \mathbf{h})}}{\cancel{\frac{1}{Z}} \sum_{\mathbf{h}} e^{-E(\mathbf{v}^{(\mathbf{d})}, \mathbf{h})}}$$

when we compute the derivative we ultimately obtain:

$$\frac{\partial}{\partial \boldsymbol{\theta}} (\ln \mathcal{L}(\mathbf{v}^{(\mathbf{d})}|\boldsymbol{\theta})) = -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}^{(\mathbf{d})}) \frac{\partial E(\mathbf{v}^{(\mathbf{d})}, \mathbf{h})}{\partial \boldsymbol{\theta}} + \sum_{\mathbf{v}^{(\mathbf{d})}, \mathbf{h}} P(\mathbf{v}^{(\mathbf{d})}, \mathbf{h}) \frac{\partial E(\mathbf{v}^{(\mathbf{d})}, \mathbf{h})}{\partial \boldsymbol{\theta}} \tag{1.5}$$

[6] Further insights on KL divergence can be found in section 2.2.2.

Therefore, the derivative of the log-likelihood with respect to the parameters $\boldsymbol{\theta}$ is equal to the difference between two expectations: the expected values of the energy function under the conditional distribution of the hidden variables given the training examples (often referred to as the positive phase), and the expected values of the energy function under the model distribution (the so called negative phase). In general, the direct computation of these sums results in an exponential computational complexity with respect to the number of variables in the MRF. To reduce this computational complexity, the expectations can be approximated using samples drawn from the corresponding distributions through Markov chain Monte Carlo (MCMC) techniques[7].

1.6 Unsupervised learning for RBMs

Since RBMs can be seen as MRFs associated with a bipartite undirected graph, we can apply the same formalism previously discussed for MRFs. As shown in the previous section, see (1.5), the gradient of the log-likelihood of an MRF can be written as the difference of two expectations. For RBMs the first term of (1.5) can be computed efficiently because it factorizes nicely thanks to the property (1.4). Instead, for the second term in (1.5) the computation remains intractable even if we try to utilize the same kind of factorization. The detailed calculations are shown in [11].

For example, given a single training example $\boldsymbol{v^d}$, the derivative (w.r.t. the weights W_{ij}) of the log-likelihood can be written as:

$$\frac{\partial}{\partial W_{ij}}(\ln \mathcal{L}(\mathbf{v^{(d)}}|\boldsymbol{\theta})) = \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v^{(d)}}) h_j v_i^{(d)} - \sum_{\mathbf{v^{(d)}}} P(\mathbf{v^{(d)}}) \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v^{(d)}}) h_j v_i^{(d)} \tag{1.6}$$

For the full training set $D = (\mathbf{v^{(1)}}, ..., \mathbf{v^{(N_D)}})$, it is possible to show that the mean of this derivative is:

$$\frac{1}{N_D} \sum_{d=1}^{N_D} \frac{\partial \ln \mathcal{L}(\mathbf{v^{(d)}}|\boldsymbol{\theta})}{\partial W_{ij}} \propto < v_i h_j >_{data} - < v_i h_j >_{model} \tag{1.7}$$

[7] More information on MCMC techniques can be found in the Appendix.

Similarly, it can be demonstrated that calculating the derivatives with respect to the bias parameters a_i and b_j also requires determining the difference between a moment from the data and the corresponding moment from the model distribution. In general, computing this latter moment is an NP-hard problem. This expectation can be approximated using samples from the model distribution, which can be obtained through methods such as Gibbs sampling[8]. However, the computational cost of this Markov chain Monte Carlo (MCMC) approach is too high for efficient learning algorithms. Therefore, common RBM learning techniques, as described in Section 1.7, incorporate additional approximations to mitigate these costs.

1.7 Different training algorithms for approximating the RBM log-likelihood gradient

1.7.1 Contrastive Divergence

The Contrastive Divergence (CD) algorithm is an MCMC-based method proposed by Geoffrey Hinton in 2002 [6] for training RBMs. Differently from Gibbs sampling, in which we have to run a Markov chain starting from a random configuration and wait until equilibrium is reached, in the CD algorithm we initialize our chain with a data sample $\mathbf{v^{(0)}}$ from the training set and after k alternating steps (typically $k = 1$) we obtain the sample $\mathbf{v^{(k)}}$. At each step t, leveraging the bipartite structure of RBMs, we can sample the hidden configuration $\mathbf{h^{(t)}}$ from $P(\mathbf{h}|\mathbf{v^{(k)}})$ and then we can sample $\mathbf{v^{(t)}}$ from $P(\mathbf{v}|\mathbf{h^{(t)}})$. As a result, for a single training example $\mathbf{v^{(0)}}$ the gradient, described in equation (1.5), with respect to the weights takes the form:

[8] More information on Gibbs sampling can be found in the Appendix.

$$CD_k(\mathbf{W}, \mathbf{v}^{(0)}) = -\sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}^{(0)}) \frac{\partial E(\mathbf{v}^{(0)}, \mathbf{h})}{\partial \mathbf{W}} + \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}^{(k)}) \frac{\partial E(\mathbf{v}^{(k)}, \mathbf{h})}{\partial \mathbf{W}} \tag{1.8}$$

$$= \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}^{(0)}) \mathbf{v}^{(0)} \mathbf{h}^T - \sum_{\mathbf{h}} P(\mathbf{h}|\mathbf{v}^{(k)}) \mathbf{v}^{(k)} \mathbf{h}^T \tag{1.9}$$

Finally, we iterate this procedure for multiple training examples to refine the model parameters further.

1.7.2 Persistent Contrastive Divergence

The CD algorithm typically suffers from limited mixing in the Gibbs sampling process because it starts the Markov chain from a training sample and does not run it for enough steps to reach equilibrium. Therefore, when performing the sampling we only explore a neighborhood of the data samples and the regions that are far away from the original data are typically never seen.
In order to address this mixing issue, Tieleman introduced in 2008 [12] the Persistent Contrastive Divergence (PCD) algorithm. The key novelty introduced by PCD is that instead of starting a new Markov chain each time we want to draw a sample, it maintains a set of "persistent" chains that are updated throughout the training process so that the initial state of the current Gibbs chain is equal to the last state obtained from the previous update step.

1.7.3 Parallel Tempering

The final algorithm for training RBMs that I will briefly introduce is called Parallel Tempering (PT) [13], also known as the "replica exchange method".
To improve the efficiency of MCMC simulations, PT algorithm runs in parallel multiple replicas $\{(\mathbf{v_r}, \mathbf{h_r}), r = 1, ..., N_R\}$ of the system, each drawn from the Gibbs distribution at a different temperature T_r:

$$P_{T_r}(\mathbf{v}, \mathbf{h}) = \frac{1}{Z_{T_r}} e^{-\frac{E(\mathbf{v},\mathbf{h})}{T_r}} \tag{1.10}$$

with: $1 = T_1 < T_2 < ... < T_{N_R}$ Notice that at high temperatures ($T_r >> 1$) the chain mixes well since the distribution tends to a uniform one. In order to make use of these well mixed chains that we find at high temperatures, at each time-step two neighbouring Gibbs chains with temperatures T_r and T_{r+1} may exchange their "particles" $(\mathbf{v_r}, \mathbf{h_r})$ and $(\mathbf{v_{r+1}}, \mathbf{h_{r+1}})$ with an exchange probability based on Metropolis ratio:

$$\min\{1, \frac{P_{T_r}(\mathbf{v_{r+1}}, \mathbf{h_{r+1}})P_{T_{r+1}}(\mathbf{v_r}, \mathbf{h_r})}{P_{T_r}(\mathbf{v_r}, \mathbf{h_r})P_{T_{r+1}}(\mathbf{v_{r+1}}, \mathbf{h_{r+1}})}\} \tag{1.11}$$

This approach enables the system to explore the energy landscape more effectively and it ultimately leads to a more efficient and robust training process for RBMs with respect to CD and PCD. In the end, the samples from the replica at $T = 1$ are used for training the RBM, as they correspond to the desired target distribution.

In the next chapter, we will explore an alternative and promising inference algorithm called Expectation Propagation (EP), which is the method employed in my thesis for training RBMs.

Chapter 2

Expectation Propagation: Theory and Fundamentals

2.1 Introduction

Expectation Propagation (EP) is a technique used in Bayesian inference for approximating unknown probability distributions with tractable ones.

Originally introduced in the field of statistical physics with the name of Expectation Consistent by Opper & Winther in their two seminal papers [14, 15] and later popularized by Thomas Minka in his doctoral thesis [16], EP aimed to address a major drawback of Bayesian methods, namely, their computational expense[1]. To accomplish this, Minka in his work unified and generalized two previous techniques: assumed-density filtering (ADF)[17] and loopy belief propagation (LBP)[18].

In the upcoming section, prior to delving into the details of EP, we will first examine some of its foundational concepts.

[1] Typically, the computation of the exact posterior requires solving high-dimensional integrals that lack an analytical solution.

2.2 Preliminaries

2.2.1 Exponential Families

EP makes extensive use of exponential families due to their unique properties and versatility in representing probability distributions.
An exponential family of distributions is a parametric set of probability distributions that can be written in the following form:

$$P(x|\boldsymbol{\theta}) = e^{\sum_{\alpha} \theta_\alpha f_\alpha(x) - \Phi(\boldsymbol{\theta})}$$

where:

- x represents the observed data
- $\boldsymbol{\theta}$ is the parameter vector that identifies the various distributions within the family
- $f_\alpha(x)$ is a function from the space of possible values of x to the real numbers.
- $\Phi(\boldsymbol{\theta})$ is the *log partition function* ensuring the normalization of $P(x|\boldsymbol{\theta})$

Interestingly, many of the most common distributions (e.g. the Gaussian distribution, the Poisson distribution and the Bernoulli distribution) can be written in exponential form.
The importance of exponential families lies in their numerous properties that simplify standard computations. For example, when we multiply or divide two exponential distributions, we generate a new distribution belonging to the same exponential family (although normalization might be compromised). Moreover, the coefficients of the resulting product or quotient distribution are simply equivalent to the addition or subtraction of the input coefficients.

2.2.2 Kullback-Leibler divergence

The Kullback-Leibler (KL) divergence quantifies the "distance" between two probability densities and it is formally defined (for discrete variables) as follows:

$$D_{KL}[P(\mathbf{x})||Q(\mathbf{x})] \doteq \sum_{\mathbf{x}} P(\mathbf{x}) log \frac{P(\mathbf{x})}{Q(\mathbf{x})}$$

where $P(\mathbf{x})$ is the target probabilistic distribution and $Q(\mathbf{x})$ the approximated one. In case the variables are continuous, one has to replace summation with integration.
The minimization of KL divergence $D_{KL}[P(\mathbf{x})||Q(\mathbf{x})]$ is one of the key passages of the EP algorithm. In particular, if the approximate distribution $Q(\mathbf{x})$ is constrained to a member of the exponential family, minimizing the KL divergence corresponds to a moment-matching problem, as shown for example in [19].

2.2.3 Assumed-density Filtering

Assumed-density filtering (ADF) is a general technique utilized not only in Bayesian networks but also in other statistical models to calculate approximate posterior distributions. ADF has been introduced independently in the fields of statistics [20], artificial intelligence [21], and control literature [22]. More precisely, the term "Assumed-density filtering" is the one used in control theory, whereas in other fields ADF is also known as "online Bayesian learning"[17] or "weak marginalization"[20].

Let us now introduce the ADF algorithm. Consider a set of observed data $D = (\mathbf{v^{(1)}}, ..., \mathbf{v^{(N_D)}})$ from which we seek to infer a hidden vector $\mathbf{x}$. We start by postulating the prior $P(\mathbf{x})$ and the likelihood $P(D|\mathbf{x})$, enabling us to compute the posterior $P(\mathbf{x}|D)$ via Bayes' theorem:

$$P(\mathbf{x}|D) = \frac{P(D|\mathbf{x})P(\mathbf{x})}{P(D)}$$

However, even when the prior $P(\mathbf{x})$ is tractable and the likelihood $P(D|\mathbf{x}) = \prod_{a=1}^{N_D} P(\mathbf{v^{(a)}}|\mathbf{x})$ is factorized (since D is a set of independent observations), the

computation of the posterior can be hard depending on the form of the likelihood functions.

In order to estimate the posterior distribution, ADF iteratively update a trial probability distribution $Q(\mathbf{x})$ based on incoming evidence. If the prior is tractable, it is reasonable to initially choose $Q(\mathbf{x})$ of the same family of the prior $P(\mathbf{x})$ or even more simply $P(\mathbf{x}) = Q(\mathbf{x})$. Considering the latter case, given the first set of data $\mathbf{v^{(1)}}$ the posterior will be modified as follows:

$$P^{a=1}(\mathbf{x}|D) = \frac{Q(\mathbf{x})P(\mathbf{v^{(1)}}|\mathbf{x})}{\int_{\mathbf{x}} Q(\mathbf{x})P(\mathbf{v^{(1)}}|\mathbf{x})}$$

The new approximated distribution $Q^{new}(\mathbf{x})$ can be computed by minimizing the KL divergence $D_{KL}[P^{a=1}||Q^{new}]$ and this, as anticipated before, corresponds to the matching of the first two moments of the distributions.

Once $Q = Q^{new}$ has been updated, the moments are *propagated*, in the sense that when we observe $\mathbf{v^{(2)}}$ and compute the partial posterior $P^{a=2}(\mathbf{x}|D) \propto Q(\mathbf{x})P(\mathbf{v^{(1)}}|\mathbf{x})$, the information about the first set of data $\mathbf{v^{(1)}}$ is absorbed in the expectations of $Q(\mathbf{x})$. The procedure continues in this way until all the N_D observations are made.

The main drawback of the ADF algorithm is that it has the undesirable property of being strongly sensitive to the order in which the data are observed. As we will see later on, EP overcomes this issue by revisiting each approximation term multiple times.

2.3 Understanding Expectation Propagation

Essentially, EP can be seen as an iterative refinement of ADF, offering the advantage of being order-independent with respect to observations. Indeed, since in ADF the approximations do not have any required order, we have the freedom to go back and refine them according to our preferences.

Let us now provide a detailed description of the EP algorithm.

Consider a set of random variables, denoted as $\mathbf{x}$, and let $P(\mathbf{x})$ represent a distribution formed by the multiplication of various "compatibility functions"

$\Psi_i(\mathbf{x})$:

$$P(\mathbf{x}) \propto \prod_i \Psi_i(\mathbf{x})$$

EP aims to provide tractable approximations to complex probability functions of this form. For instance, consider a set of N_D independent observations $D = (\mathbf{v^{(1)}}, ..., \mathbf{v^{(N_D)}})$ of an unobserved random variable $\mathbf{x}$ with a prior distribution $P_0(\mathbf{x})$. According to Bayes' rule, the posterior distribution over $\mathbf{x}$ is given by:

$$P(\mathbf{x}|D) = \frac{P_0(\mathbf{x}) \prod_{i=1}^{N_D} P_i(\mathbf{v^{(i)}}|\mathbf{x})}{P(D)} \propto P_0(\mathbf{x}) \prod_{i=1}^{N_D} P_i(\mathbf{v^{(i)}}|\mathbf{x}) = P_0(\mathbf{x}) \prod_{i=1}^{N_D} \Psi_i(\mathbf{x})$$

To make this posterior distribution tractable, EP approximates each compatibility function $\Psi_i(\mathbf{x})$ with an appropriately chosen function $\phi_i(\mathbf{x})$ from the exponential family. Thus, the approximate posterior distribution is represented as:

$$Q(\mathbf{x}; \boldsymbol{\theta}) = \frac{1}{Z_Q} P_0(\mathbf{x}) \prod_{i=1}^{N_D} \phi_i(\mathbf{x})$$

where Z_Q is a normalization factor and $\boldsymbol{\theta}$ is the set of parameters identifying the approximated distribution.
In order to determine the functions $\phi_i(\mathbf{x})$ that provide the closest approximation to the factors $\Psi_i(\mathbf{x})$, we iterate through these steps until all functions $\phi_i(\mathbf{x})$ have converged:

- Select the factor $\phi_i(\mathbf{x})$ to update
- Compute the so called *cavity distribution* of the i-th variable, which is given by the product of all the Gaussian approximating factors, except for the i-th factor:
$$Q^{\setminus i}(\mathbf{x}; \boldsymbol{\theta}^{\setminus i}) \propto \frac{Q(\mathbf{x}; \boldsymbol{\theta})}{\phi_i(\mathbf{x})}$$
- Compute the *tilted* (also known as *leave-one out-distribution*) of the i-th variable, which is the product of all the Gaussian approximating factors

except for the i-th variable which is replaced by the exact prior $\Psi_i(\mathbf{x})$. Therefore, the tilted distribution is simply obtained by multiplying the cavity distribution for the i-th variable by the i-th exact prior:

$$Q^{(i)}(\mathbf{x};\boldsymbol{\theta}^{\setminus i}) = Q^{\setminus i}(\mathbf{x};\boldsymbol{\theta}^{\setminus i})\Psi_i(\mathbf{x})$$

- Find the parameters $\boldsymbol{\theta}^*$ minimizing the KL divergence $D_{KL}[Q^{(i)}(\mathbf{x};\boldsymbol{\theta}^{\setminus i})||Q(\mathbf{x},\boldsymbol{\theta})]$ through moment matching of the two distributions:

$$\begin{cases} < x_i >_{Q^{(i)}} = < x_i >_Q \\ < x_i^2 >_{Q^{(i)}} = < x_i^2 >_Q \end{cases} \tag{2.1}$$

- Update $\phi_i(\mathbf{x})$ as:

$$\phi_i(\mathbf{x}) = Z_i \frac{Q(\mathbf{x},\boldsymbol{\theta}^*)}{Q^{\setminus i}(\mathbf{x},\boldsymbol{\theta}^{\setminus i})}$$

Once all the approximated factors have been computed, we get:

$$P(\mathbf{x}) \approx Q(\mathbf{x};\boldsymbol{\theta}) = \frac{1}{Z_Q} P_0(\mathbf{x}) \prod_{i=1}^{N_D} \phi_i(\mathbf{x})$$

Chapter 3

Experimental Evaluation and Results

In this chapter, I will show the results obtained when implementing the EP algorithm on RBMs specifically trained on the MNIST database. Before describing the implementation, I will provide an overview of the MNIST dataset and the mathematical derivation for the specific model under investigation.

3.1 The MNIST Database

The MNIST (Modified National Institute of Standards and Technology) database [23] is a large database of 70,000 handwritten labeled digits (between 0 and 9), where each digit has been size-normalized and centered in a fixed-size 28x28 pixel grayscale image. Here, L=28 denotes the dimension of each side of the square image matrix. The dataset is split into 60,000 images for training and 10,000 images for testing. Each image is represented as a 28x28 matrix whose elements can assume values ranging from 0 (black) to 1 (white). Because of its simplicity, yet with some variability, the MNIST dataset is well-suited for testing the efficacy of various machine learning models, including RBMs.

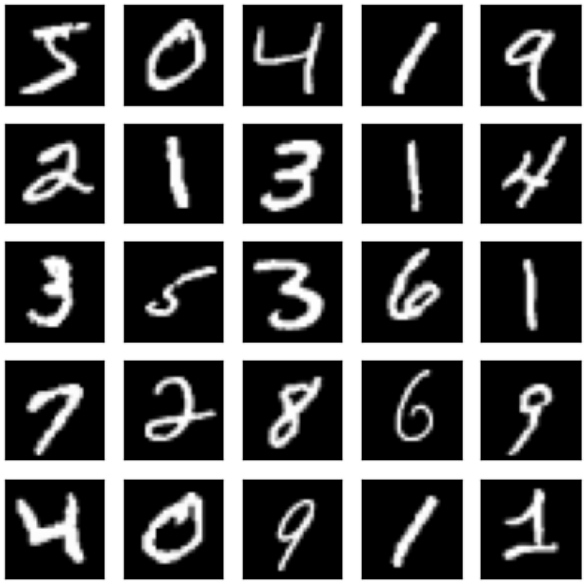


Figure 3.1: Sample of handwritten digits from the MNIST dataset

3.2 Training the RBM

Let us start by considering an RBM trained on the MNIST dataset consisting of M Gaussian hidden units $\mathbf{h} = (h_1, ..., h_M)$ and N visible units $\mathbf{v} = (v_1, ..., v_N)$. Each hidden unit is initialized with a univariate Gaussian prior $\mathcal{N}(h_\mu; m_\mu, \sigma_\mu)$, where m_μ is the mean and σ_μ the variance. Two different initialization methods were employed during the training:

- $m_\mu = 0, \sigma_\mu = 1 \; \forall \mu$ referred to as "Hopfield initialization"
- $m_\mu = \text{rand}(), \sigma_\mu = 1 \; \forall \mu$ referred to as "random initialization", where rand() is a function that generates random values uniformly distributed between 0 and 1.

In contrast, each visible unit v_i is modeled using a two-component Gaussian mixture distribution $\psi_i(v_i)$ of the form:

$$\psi_i\left(v_i\right) \propto \rho e^{-\frac{\left(v_i-\mu_1\right)^2}{2\sigma^2}} + (1-\rho)\, e^{-\frac{\left(v_i-\mu_2\right)^2}{2\sigma^2}} \tag{3.1}$$

where:

- ρ is the probability associated with the first Gaussian component of the mixture
- μ_1 and μ_2 are, respectively, the means of the first and the second Gaussian components
- σ^2 is the variance shared by both Gaussian components

The visible units of the RBM correspond to the pixels of our images, while the hidden units attempt to model the interactions among the pixels. The advantage of considering Gaussian priors is that they are able to capture the continuous variations in pixel intensities found in the MNIST dataset. Moreover, when working with Gaussian distributions, operations such as marginalization and conditioning, can be performed analytically and efficiently. Furthermore, Gaussian priors allow for more stable gradient ascent during training because they provide a continuous and differentiable activation function, which helps in maintaining smooth gradients and can lead to faster convergence and better performance when trying to capture the underlying data distribution. Then, the joint distribution of the visible and hidden units reads:

$$P(\boldsymbol{v}, \boldsymbol{h}|\mathbf{W}) = \frac{1}{Z(\mathbf{W})} e^{\sum_{i,\mu} v_i W_{i,\mu} h_\mu} \prod_i \psi_i(v_i) \prod_\mu \mathcal{N}(h_\mu) \tag{3.2}$$

where, $Z(\mathbf{W}) = \int d^N \boldsymbol{v} P\left(\boldsymbol{v}\right) \int d^M \boldsymbol{h}\, e^{\sum_{i,\mu} v_i W_{i\mu} h_\mu - \frac{\left(h_\mu - m_\mu\right)^2}{2\sigma_h^2}}$ is the partition function ensuring the normalization of the joint probability distribution.

As illustrated in Chapter 1, our aim is to find the weights $W_{i\mu}$ maximizing the likelihood (which is the same of maximizing the log-likelihood l) so that we can update the weights according to:

$$W_{i,\mu}^{t+1} = W_{i,\mu}^t + \eta \left.\frac{\partial l\left(\mathcal{D}; \mathbf{W}\right)}{\partial W_{i\mu}}\right|_t \tag{3.3}$$

Let us consider a set of N_D images $D = (\mathbf{v^{(1)}}, ..., \mathbf{v^{(N_D)}})$ then the likelihood is given by the following expression:

$$\mathcal{L}(\mathcal{D}; \mathbf{W}) = \prod_m P(\boldsymbol{v}^{(m)}|\mathbf{W}) = \prod_m \int d^M \boldsymbol{h}\, P(\boldsymbol{v}^{(m)}, \boldsymbol{h}|\mathbf{W}) = \\ = \frac{1}{Z(\mathbf{W})} \prod_m \int d^M \boldsymbol{h} e^{\sum_{i,\mu} v_i^{(m)} W_{i\mu} h_\mu - \frac{(h_\mu - m_\mu)^2}{2\sigma_h^2}} \tag{3.4}$$

Note that in the last step, we discarded the product $\prod_{i,m} \psi_i(v_i^{(m)})$ since it does not depend on $\mathbf{W}$ and therefore it is not involved in the likelihood maximization. Then, by rewriting the expression in the exponent in vector form and manipulating it, one obtains:

$$\sum_{i,\mu} v_i^{(m)} W_{i,\mu} h_\mu - \sum_\mu \frac{(h_\mu - m_\mu)^2}{2\sigma_\mu^2} = \boldsymbol{v}^{(m)^T} \mathbf{W} \boldsymbol{h} - \frac{1}{2} (\boldsymbol{h} - \boldsymbol{m})^T \boldsymbol{\Sigma}_h^{-1} (\boldsymbol{h} - \boldsymbol{m}) = \\ = -\frac{1}{2} \boldsymbol{h}^T \boldsymbol{\Sigma}_h^{-1} \boldsymbol{h} + (\boldsymbol{v}^{(m)^T} \mathbf{W} + \boldsymbol{m}^T \boldsymbol{\Sigma}_h^{-1}) \boldsymbol{h} \tag{3.5}$$

Where:

$$\boldsymbol{\Sigma}_h^{-1} = \begin{pmatrix} \frac{1}{\sigma_1^2} & 0 & \dots & 0 \\ 0 & \dots & & \dots \\ \dots & & & 0 \\ 0 & \dots & 0 & \frac{1}{\sigma_M^2} \end{pmatrix} \tag{3.6}$$

is the $M \times M$ precision matrix for the vector $\mathbf{h}$ of the hidden units. Eventually, it is possible to show that the likelihood can be reexpressed in the following form:

$$\mathcal{L}(\mathcal{D}; \mathbf{W}) = \frac{1}{Z(\mathbf{W})} \prod_m e^{\frac{1}{2} \boldsymbol{v}^{(m)^T} \mathbf{W} \boldsymbol{\Sigma}_h \mathbf{W}^T \boldsymbol{v}^{(m)} + \boldsymbol{m}^T \mathbf{W}^T \boldsymbol{v}^{(m)}} \tag{3.7}$$

while the partition function can be rewritten as:

$$Z(\mathbf{W}) = \int d^N \boldsymbol{v} P(\boldsymbol{v})\, e^{\frac{1}{2} \boldsymbol{v}^T \mathbf{W} \boldsymbol{\Sigma}_h \mathbf{W}^T \boldsymbol{v} + \boldsymbol{m}^T \mathbf{W}^T \boldsymbol{v}} \tag{3.8}$$

Then, the log-likelihood reads:

$$\begin{aligned} l(\mathcal{D};\mathbf{W}) \propto \log \mathcal{L}(\mathcal{D};\mathbf{W}) = \\ &= \frac{1}{N_D}\sum_m \left[\frac{1}{2}\boldsymbol{v}^{(m)^T}\mathbf{W}\boldsymbol{\Sigma}_h\mathbf{W}^T\boldsymbol{v}^{(m)} + \boldsymbol{m}^T\mathbf{W}^T\boldsymbol{v}^{(m)}\right] - \log Z\left(\mathbf{W}\right) = \\ &= \frac{1}{2}\sum_{i,j}\sum_{\mu,\nu} f_{ij}W_{i\mu}\Sigma^h_{\mu\nu}W_{j\nu} + \sum_\mu\sum_i m_\mu W_{i\mu}f_i - \log Z\left(\mathbf{W}\right) \\ &= \frac{1}{2}\sum_{i,j}\sum_{\mu} f_i W_{i\mu}\Sigma^h_{\mu\mu}W_{j\mu}f_j + \sum_\mu\sum_i m_\mu W_{i\mu}f_i - \log Z\left(\mathbf{W}\right) \end{aligned} \tag{3.9}$$

where

$$f_i = \frac{1}{N_D}\sum_m v_i^{(m)} = \langle v_i\rangle_{\mathcal{D}} \tag{3.10}$$

$$f_{ij} = \frac{1}{N_D}\sum_m v_i^{(m)}v_j^{(m)} = \langle v_i v_j\rangle_{\mathcal{D}} \tag{3.11}$$

represent the empirical frequency counts for the i-th visible variable and for both the i-th and j-th visible variables, respectively. Notice that in the last step we simply exploited the fact that $\Sigma^h_{\mu\nu} = \frac{1}{\sigma^2_\mu}\delta_{\nu,\mu}$.
Finally, maximizing with respect to the weights we get:

$$\frac{\partial l\left(\mathcal{D};\mathbf{W}\right)}{\partial W_{i\mu}} = \sum_j \Sigma^h_{\mu\mu}\left[\langle v_i v_j\rangle_{\mathcal{D}} - \langle v_i v_j\rangle_{EP}\right]W_{j\mu} + m_\mu\left[\langle v_i\rangle_{\mathcal{D}} - \langle v_i\rangle_{EP}\right] \tag{3.12}$$

Once again, the derivative of the log-likelihood with respect to the weights can be expressed as the difference between two moments: the moments of the data and the moments from the model distribution, with the latter being calculated using EP. Therefore, following the EP procedure detailed in 2.3, we approximate the factorized target distribution:

$$P(\boldsymbol{v}|\mathbf{W}) = \int d\mathbf{h}P(\boldsymbol{v},\boldsymbol{h}|\mathbf{W}) = \frac{1}{Z_v}e^{\frac{1}{2}\boldsymbol{v}^T\mathbf{W}\boldsymbol{\Sigma}_h\mathbf{W}^T\boldsymbol{v}+\boldsymbol{m}^T\mathbf{W}^T\boldsymbol{v}}\prod_i \psi_i(v_i) \tag{3.13}$$

using an approximating distribution $Q(\mathbf{v})$ of the form:

$$Q(\mathbf{v}) = \frac{1}{Z_Q}e^{\frac{1}{2}\boldsymbol{v}^T\mathbf{W}\boldsymbol{\Sigma}_h\mathbf{W}^T\boldsymbol{v}+\boldsymbol{m}^T\mathbf{W}^T\boldsymbol{v}}\prod_i \phi_i(v_i) \tag{3.14}$$

where each $\phi_i(v_i)$ approximates the corresponding exact prior factor $\psi_i(v_i)$. Given the Gaussian nature of the priors, the optimal choice for $\phi_i(v_i)$ is Gaussian, specifically: $\phi_i(v_i) \sim \mathcal{N}(a_i, b_i)$.

Then, in order to compute the approximated marginal distribution $Q(v_i)$ for the i-th visible variable, we need to express the marginal tilted distribution

$$Q^{(i)}(v_i) \propto e^{\frac{1}{2}\boldsymbol{v}^T \mathbf{W}\boldsymbol{\Sigma}_h \mathbf{W}^T \boldsymbol{v} + \boldsymbol{m}^T \mathbf{W}^T \boldsymbol{v}} \psi_i(v_i) \prod_{j \neq i} \phi_j(v_j) \tag{3.15}$$

in terms of the cavity distribution

$$Q^{\setminus i}(v_i) \propto \frac{Q(\mathbf{v})}{\phi_i(v_i)} \propto e^{\frac{1}{2}\boldsymbol{v}^T \mathbf{W}\boldsymbol{\Sigma}_h \mathbf{W}^T \boldsymbol{v} + \boldsymbol{m}^T \mathbf{W}^T \boldsymbol{v}} \prod_{j \neq i} \phi_j(v_j) \tag{3.16}$$

Then, we obtain the desired form:

$$Q^{(i)}(v_i) \propto \psi_i(v_i) Q^{\setminus i}(v_i) \propto \psi_i(v_i)\ e^{-\frac{(v_i - \mu_i^v)^2}{2\Sigma_{ii}^v}} \tag{3.17}$$

where, μ_i^v, Σ_{ii}^v are the mean and covariance of the cavity marginal $Q^{\setminus i}(v_i)$, respectively. Then, the fully approximated marginal distribution for the i-th visible variable is given by:

$$Q_i(v_i) \propto Q^{\setminus i}(v_i)\, \phi_i(v_i) \tag{3.18}$$

Subsequently, it is possible to show that the mean $\mathbf{d}$ and covariance matrix $\mathbf{C_v}$ of the fully approximated distribution $Q(\mathbf{v})$, can be expressed in terms of the mean $\mathbf{a}_v$ and the precision matrix $\boldsymbol{D}_v^{-1}$ of the multivariate Gaussian distribution $\phi(\mathbf{v}) = (\phi_1(v_1), ..., \phi_N(v_N)$ as follows:

$$\boldsymbol{C}_v^{-1} = \mathbf{D}_v^{-1} - \mathbf{W}\boldsymbol{\Sigma}_h \mathbf{W}^T \tag{3.19}$$

$$\boldsymbol{d} = \boldsymbol{C}_v \left(\mathbf{D}_v^{-1} \boldsymbol{a}_v + \mathbf{W}\boldsymbol{m}\right) \tag{3.20}$$

where, $\boldsymbol{D}_v^{-1}$ is given by:

$$\boldsymbol{D}_v^{-1} = \begin{pmatrix} \frac{1}{b_1} & 0 & \dots & 0 \\ 0 & \dots & & \dots \\ \dots & & & 0 \\ 0 & \dots & 0 & \frac{1}{b_N} \end{pmatrix} \tag{3.21}$$

Eq. 3.19 is the starting point of our algorithm. Once $\mathbf{d}$ and $\mathbf{C_v}$ have been computed, we can incorporate them in the subsequent equations, determining

the parameters of the cavity marginal $Q^{\setminus i}(v_i)$ in terms of the parameters of the fully Gaussian distribution:

$$\mu_i^v = \frac{d_i - \frac{a_i C_v(i,i)}{b_i}}{1 - \frac{C_v(i,i)}{b_i}} \tag{3.22}$$

$$\Sigma_{ii}^v = \frac{C_v(i,i)}{1 - \frac{C_v(i,i)}{b_i}} \tag{3.23}$$

At this point, we proceed to calculate the moments and the partition function for the tilted distribution:

$$\langle v_i \rangle_{Q^{(i)}} = \frac{1}{Z_{Q^{(i)}}} \left[\rho m_1 e^{-\frac{(\mu_i - \mu_1)^2}{2(\Sigma_{ii} + \sigma^2)}} + (1-\rho)\, m_2 e^{-\frac{(\mu_i - \mu_2)^2}{2(\Sigma_{ii} + \sigma^2)}} \right] \tag{3.24}$$

$$\mathrm{Var}(v_i) = \frac{1}{Z_{Q^{(i)}}} \left[\rho \Sigma_1 e^{-\frac{(\mu_i - \mu_1)^2}{2(\Sigma_{ii} + \sigma^2)}} + (1-\rho)\, \Sigma_2 e^{-\frac{(\mu_i - \mu_2)^2}{2(\Sigma_{ii} + \sigma^2)}} \right] \tag{3.25}$$

$$Z_{Q^{(i)}} = \rho e^{-\frac{(\mu_i - \mu_1)^2}{2(\Sigma_{ii} + \sigma^2)}} + (1-\rho)\, e^{-\frac{(\mu_i - \mu_2)^2}{2(\Sigma_{ii} + \sigma^2)}} \tag{3.26}$$

Once all these quantities have been calculated, it is possible to compute the average a_i and the variance b_i of the i-th Gaussian factor $\phi_i(v_i)$, as follows:

$$b_i = \left(\frac{1}{\langle v_i^2 \rangle_{Q^{(i)}} - \langle v_i \rangle_{Q^{(i)}}^2} - \frac{1}{\Sigma_{ii}} \right)^{-1} \tag{3.27}$$

$$a_i = b_i \left[\langle v_i \rangle_{Q^{(i)}} \left(\frac{1}{b_i} + \frac{1}{\Sigma_{ii}} \right) - \frac{\mu_i}{\Sigma_{ii}} \right] \tag{3.28}$$

Finally, the desired EP moments we were looking for are given by:

$$\langle v_i \rangle_{EP} = \mu_i^v \tag{3.29}$$

$$\langle v_i v_j \rangle_{EP} \propto \Sigma_{ij}^v + \mu_i^v \mu_j^v \tag{3.30}$$

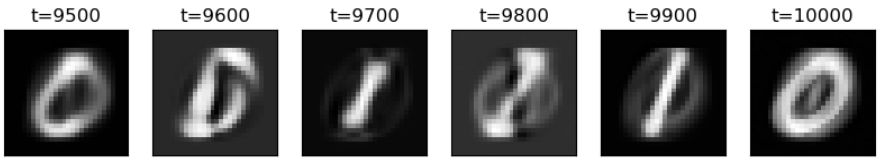
The procedure can be iterated by inserting the updated values of a_i and b_i into Eq. 3.19 for computing the new values of $\langle v_i \rangle_{EP}$ and $\langle v_i v_j \rangle_{EP}$ until convergence. These moments are then plugged into Eq. 3.12 and the weights are updated according to Eq. 3.3 for the desired number of steps.

3.3 Results of the training

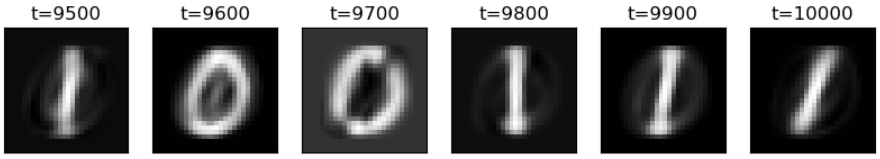
In this section, I will present a detailed analysis of all the results obtained from training an RBM on the MNIST dataset using EP. Initially, we will consider the case of a single copy, where the term "copy" refers to a unique initialization of the parameters **a** and **b** used in EP. Subsequently, we will utilize and then combine the results coming from multiple independent copies (with K being the number of copies), each initialized differently, to more effectively capture the complexity of the dataset.

3.3.1 Case of a single copy (K=1)

By implementing the mathematical framework outlined in 3.2, it is possible to compute the mean vector **d** of the distribution $Q(\mathbf{v})$, which approximates the target model distribution. This vector is reshaped into an $L \times L$ matrix for plotting, revealing the digits learned by the algorithm. For instance, using a single copy (K=1), for training images of size 28×28, with $M = 100$ hidden units, the learning process produced the following images:



(a) $m_h^{(t=0)} = \text{rand(M)}$



(b) $m_h^{(t=0)} = \text{zeros(M)}$ (Hopfield model)

Figure 3.2: Multiple 28×28 images learned with $K = 1, M = 100, T = 10000$ at different time steps. (a) corresponds to the images learned with initialization $m_h^{(t=0)} = \text{rand(M)}$, while (b) shows the images initialized with $m_h^{(t=0)} = \text{zeros(M)}$.

Regrettably, as shown by the images above, when training an RBM with a single copy, the model converges to single different digit types (0 or 1) at various epochs, failing to capture the intricate variations within these classes[1]. To assess quantitatively the accuracy of the estimations, the following parameters were employed:

1. The Pearson correlation ρ_{av} between the mean $\langle v_i \rangle_{\mathcal{D}}$ of each visible unit and the mean $\langle v_i \rangle_{EP}$ of the Gaussian mixture model computed by means of EP.

2. The Pearson correlation ρ_{sec} between the expectation value of the product $\langle v_i v_j \rangle_{\mathcal{D}}$ between the i-th and the j-th visible units and the expectation value $\langle v_i v_j \rangle_{EP}$ of the Gaussian mixture model computed by means of EP.

A reasonable approximation should yield correlation coefficients that are nearly

[1] As we will see in the following section, the correlation ρ_{sec}, a fundamental parameter to estimate the accuracy of the tests, oscillates too much in the case of a single copy.

equal to one. For $K = 1$, the correlations obtained are unsatisfactory and oscillate too much between subsequent epochs:

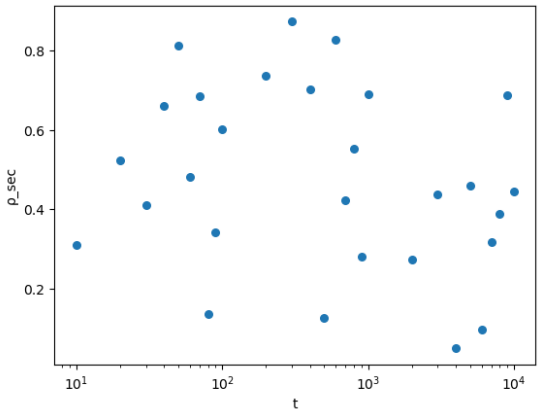
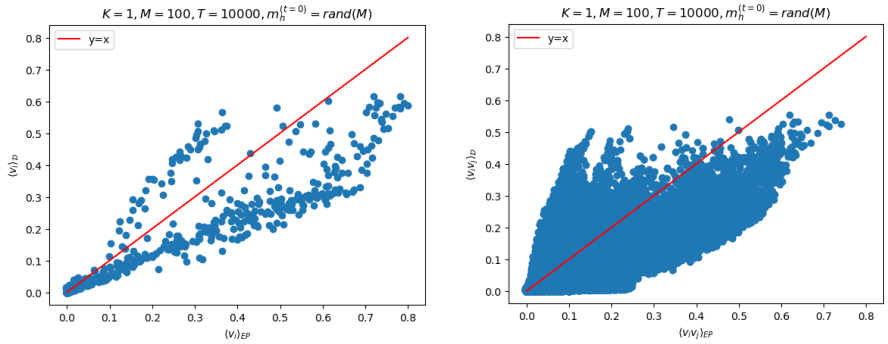


Figure 3.3: Temporal evolution of ρ_{sec}



(a) ρ_{av} (b) ρ_{sec}

Figure 3.4: Study of correlations in the case $K = 1$ for images of size 28×28 learned with $M = 100, T = 10000$ and initialized by $m_h^{(0)} = \text{rand(M)}$.

3.3.2 Leveraging additional independent copies

One way to overcome the drawbacks of using a single copy is to run the algorithm for K independent copies of the approximation, each one differently initialized according to:

```
a_repl = [randn(N) for _ in 1:K]
b_repl = [rand(N) for _ in 1:K]
```

In this case, the full distribution is simply given by the Gaussian mixture of all the K independent Gaussian distributions:

$$Q(\mathbf{v}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}^{(k)}(\mathbf{v}|\mathbf{d}^{(k)}, \mathbf{C_v}^{(k)}) \tag{3.31}$$

where π_k denotes the probability associated with the k-th independent copy and must satisfy: $\sum_{k=1}^{K} \pi_k = 1$ and $\pi_k \geq 0$. By assuming that each copy is equally probable we can simply write $\pi_k = \frac{1}{K} \, \forall k$. In Ch. 4, we will relax the assumption of equal probabilities and introduce a more refined approach where the probabilities π_k will be written in terms of the entropy of the Gaussian distributions.

Finally, by integrating Gaussian mixture techniques into the training algorithm, we can compute:

$$< v_i >_{\text{Mixture}} = \sum_{k=1}^{K} \pi_k < v_i >_k \tag{3.32}$$

$$< v_i v_j >_{\text{Mixture}} = \sum_{k=1}^{K} \pi_k < v_i v_j >_k \tag{3.33}$$

This approach enables the model to capture with more accuracy the subtle discrepancies among the handwritten digits. For instance, by setting the initial condition $m_h^{(t=0)} = \text{rand(M)}$ and considering $K = 100$ independent copies, the following results were obtained (see Figs. 3.5, 3.6, 3.7).

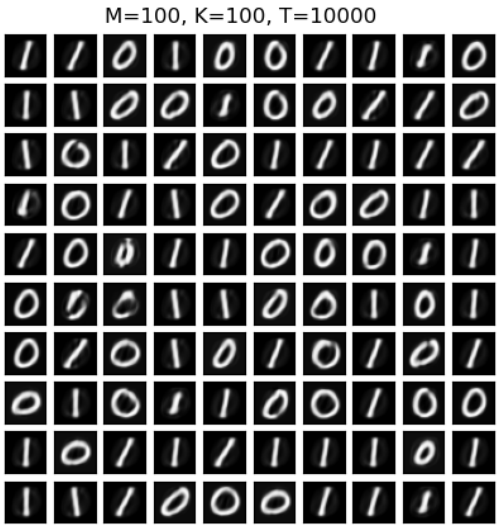


Figure 3.5: $K = 100$ images of size 28×28 learned with $M = 100, T = 10000$ initialized by $m_h^{(0)} = \text{rand(M)}$.

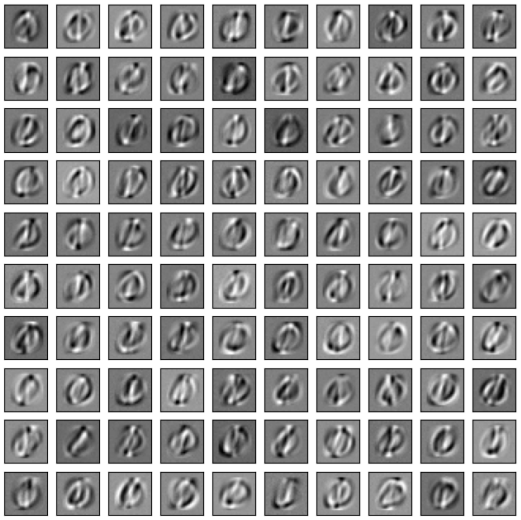


Figure 3.6: Weights W learned by each of the $K = 100$ independent copies, using images of size 28×28, with $M = 100$ and $T = 10000$, initialized by $m_h^{(t=0)} = \text{rand(M)}$.

Finally, the Gaussian mixture computation yielded the following result:

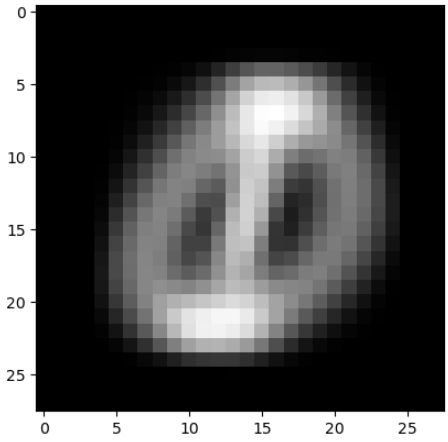
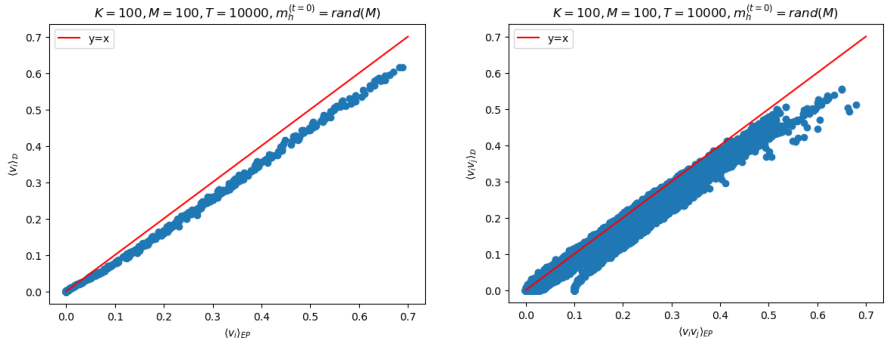


Figure 3.7: Gaussian mixture of K=100 independent copies of size 28×28 learned with $M = 100, T = 10000$ initialized by $m_h^{(0)} = \text{rand(M)}$.

As shown in the image above, it is clear that the algorithm now demonstrates enhanced accuracy in capturing simultaneously the digit 1 and the digit 0. This observation is further supported by the results obtained for the correlation coefficients depicted below:



(a) ρ_{av} (b) ρ_{sec}

Figure 3.8: Study of correlations in the case $K = 100$ for images of size 28×28 learned with $M = 100, T = 10000$ and initialized by $m_h^{(0)} = \text{rand(M)}$.

3.3.3 Max Gradient and Correlation Analysis for different values of M and K

Let us now investigate the model by examining the behavior of the maximum gradient, defined as max_grad = $\max_{i,\mu} \left| \frac{\partial l(\mathcal{D};\mathbf{W})}{\partial W_{i\mu}} \right|$, along with the parameters ρ_{av} and ρ_{sec} across various combinations of M and K values. Understanding the behavior of these parameters during training is crucial for optimizing the training process and gaining insights into the convergence dynamics and numerical stability of the RBM trained using EP.

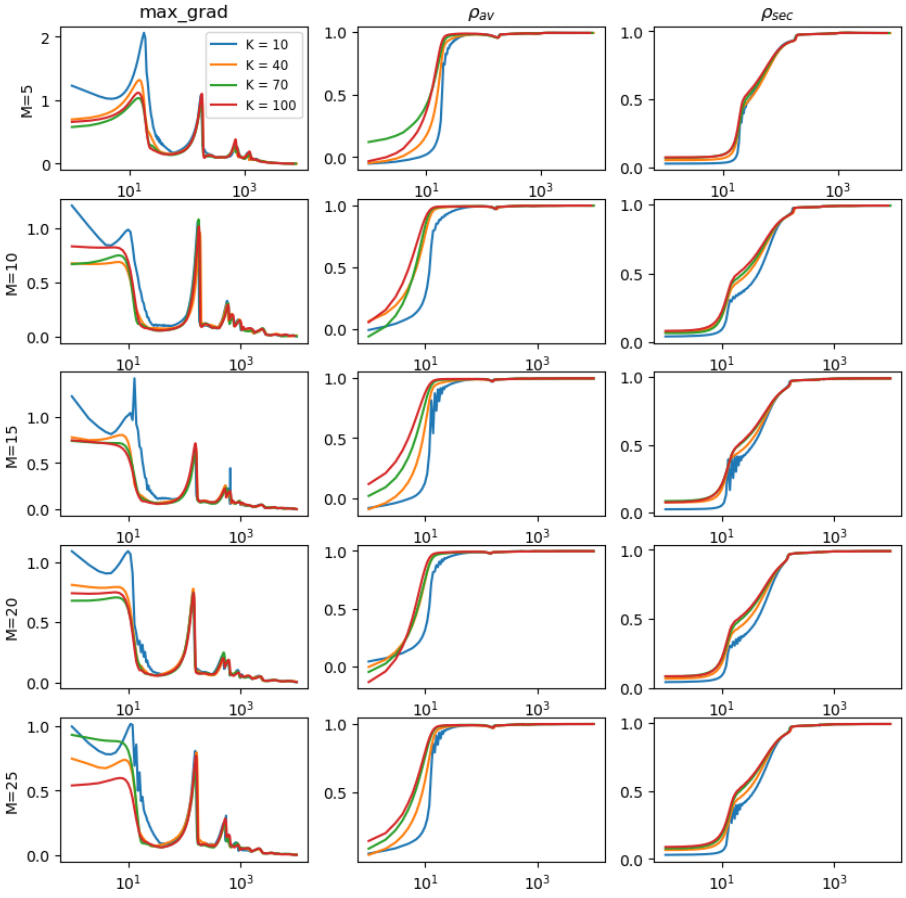


Figure 3.9: Variation of maximum gradient, ρ_{av} and ρ_{sec} for M=5,10,15,20,25 and K=10,40,70,100 in the case of 14×14 resized images and $m_h^{(t=0)} = \text{rand(M)}$.

As shown above, even a small number of copies (K) is sufficient to achieve good dynamics. However, with $K = 10$ independent copies, there is a more pronounced peak in the maximum gradient around $t = 10$, along with higher fluctuations for the correlation coefficients. This indicates greater instability during the training process. Indeed, large gradients can lead to unstable updates, causing the parameters to change too drastically between different iterations. Moreover, we can also notice that even with a small number of hidden units (M), the model performs well, effectively capturing and learning the underlying patterns in the data. We can further explore the dynamics by considering the effects of keeping K fixed while varying M (see Fig. 3.10), and vice versa (Fig. 3.11)

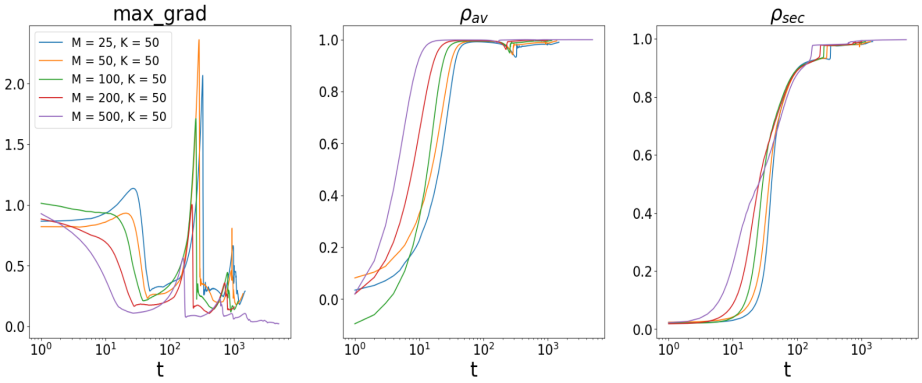


Figure 3.10: Variation of maximum gradient, ρ_{av} and ρ_{sec} for M=25,50,100,200 and fixed K=50 in the case of 14×14 resized images and $m_h^{(t=0)} = \text{rand(M)}$. The algorithm terminated upon reaching $\rho_{av}, \rho_{sec} \geq 0.99$.

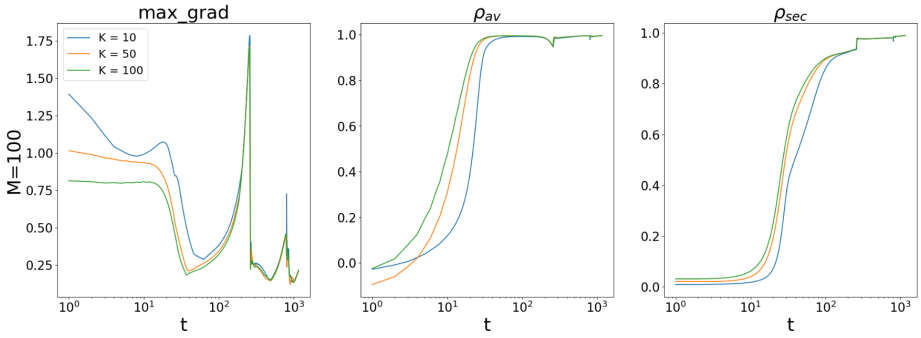


Figure 3.11: Variation of maximum gradient, ρ_{av} and ρ_{sec} for K=10,50,100 and fixed M=100 in the case of 14×14 resized images and $m_h^{(t=0)} = \text{rand(M)}$. The algorithm terminated upon reaching $\rho_{av}, \rho_{sec} \geq 0.99$.

3.4 Clustering

In light of the results presented in 3.3.3, we can now select the regimes that ensure efficient training without excessive computational costs, upon which we will apply and test clustering techniques. The idea is to exploit the mixture model obtained to classify data and get the labels of the dataset.

3.4.1 Introduction to Clustering

In the realm of unsupervised learning, clustering essentially consists in grouping a set of unlabeled examples in such a way that objects in the same group, or cluster, are more similar (according to a specific criterion chosen) to each other with respect to those in other groups.

There exist various clustering algorithms that exploit different distance measures to estimate the similarity between the examples. Each method carries its own set of strengths and weaknesses, depending on the specific application context.

3.4.2 Results from Clustering

From now on, all results presented have been obtained using the system at convergence, where both ρ_{av} and ρ_{sec} are greater than or equal to 0.99. The clustering procedure utilized can be divided into two major steps:

1. For each digit of the dataset compute its likelihood given the parameters of a specific copy k learned by EP at convergence. Thereafter, each digit is associated with the EP copy which maximizes the likelihood (basically we have found the closer copy to the digit under investigation). Once this first step of clustering is completed, we are left with thousands of images clustered into K' sets, depending on the copies. For example, in the case of 14×14 images with $M = 100, K = 70, T = 10000$ in the Hopfield case one obtains:

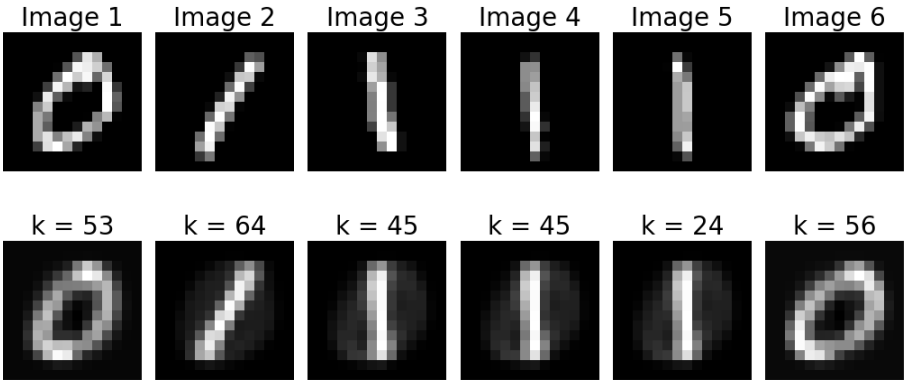


Figure 3.12: 14×14 images with their corresponding most similar EP copy found, in the case of $M = 100, K = 70, T = 10000, m_i^{(t=0)} = \text{zeros(M)}$.

2. We can further cluster by grouping together all similar independent EP copies. Additionally, we will also test the capability of the algorithm to distinguish between slanted and upright 1s, and between larger or narrower 0s from the standard ones.

The algorithm employed for this second type of clustering was K-medoids which works by identifying a set of N_M data points, called "medoids," in such

a way that the overall distance between each data point and its nearest medoid is minimal.

To quantify the distance between two copies, the normalized scalar product between the averages calculated by EP was utilized. This yields a $K \times K$ matrix, commonly referred to as the "similarity matrix". Then, by simply subtracting this matrix from the identity matrix one recovers the so-called "distance matrix" which serves as input for the k-medoids algorithm. The other required parameter for the K-medoids clustering is the number of clusters into which we want to partition the dataset.

To differentiate between just 0s and 1s, we considered two clusters ($N_M = 2$). Then, in this case, for 14×14 images with $M = 100, K = 70, T = 10000, m_h^{(t=0)} = \text{zeros(M)}$, the following results were obtained:

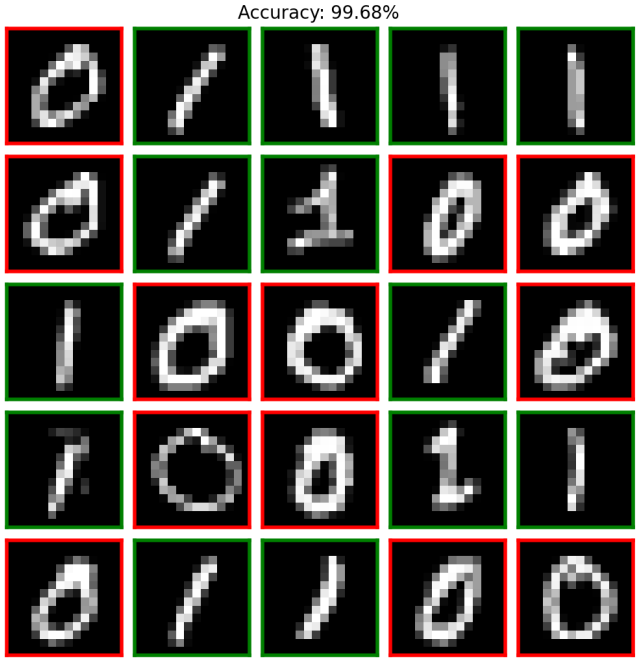


Figure 3.13: Clustering results for 14×14 images of digits 0 and 1, partitioned into $N_M = 2$ clusters (labeled in green and red) with $M = 100, K = 70, T = 10000, m_h^{(t=0)} = \text{zeros(M)}$. The grid shows 5x5 examples of the performed clustering.

As shown in the examples above, the applied clustering method effectively distinguishes between 0s and 1s, achieving an accuracy of 99.68%. Accuracy, in this context, refers to the percentage of correct classifications relative to the total number of instances.

Additionally, to visually assess the algorithm's performance across the entire MNIST dataset, we employ a confusion matrix, shown in Fig.3.14. Each cell in the matrix provides the count of data points where the model's predicted label corresponds to the true label. The diagonal terms represent the correctly classified instances, which are the cases where the predicted label matches the true label. Instead the off-diagonal terms represent the misclassified instances. The color intensity of each cell ranges from dark purple (low counts) to bright yellow (high counts), providing a visual cue of the model's prediction accuracy for each class.

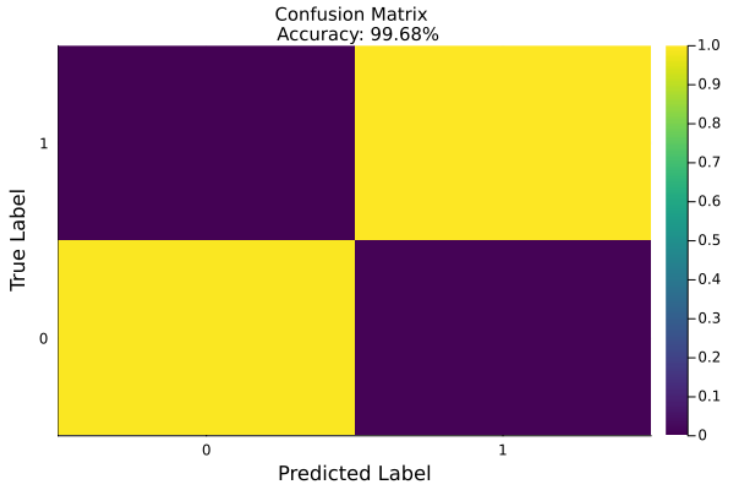


Figure 3.14: Confusion matrix for 2 clusters, based on 14×14 images, with $M = 100, K = 70, T = 10000, m_h^{(t=0)} = \text{zeros(M)}$. The color intensity represents the number of instances in each category, with darker colors indicating fewer instances and brighter colors indicating more instances.

We can further investigate whether the clustering algorithm is also capable of capturing the subtle differences between slanted 1s and upright 1s, as well as be-

tween 0s of varying sizes, with some being larger and others narrower. Through various experiments, it was observed that five clusters ($N_M = 5$) yielded the best results, as illustrated in 3.15.

Figure 3.15: Results obtained by partitioning EP copies into 5 clusters for 14×14 images with $M = 100, K = 70, T = 10000, m_h^{(t=0)} = \text{zeros(M)}$.

However, the clustering is still not optimal, since there are certain 1s featuring a bar below, look for example at the 1s in position $(3, 2)$ and $(4, 4)$, which are grouped together with the 1s without this lower bar. Additionally, the qualitative differences between the blue and red 0s are difficult to understand by looking at the pictures only.

Regrettably, the case with "rand()" initialization does not show significant improvements compared to the Hopfield case, and the model still fails to distinguish 1s with a bar below, despite the generally good clustering results:

Figure 3.16: Results obtained by partitioning EP copies into 5 clusters for 14×14 images with $M = 100, K = 70, T = 10000, m_h^{(t=0)} = \text{rand(M)}$.

3.4.3 Adding more digits

Let us now have a look at the algorithm's performance when considering more digits, for example in the case of 0, 1, 2, 3 and 4, all together.

In the case of $K = 100$ independent copies, $M = 100$ hidden units and $T = 10000$ epochs, the following images were learned by each single copy:

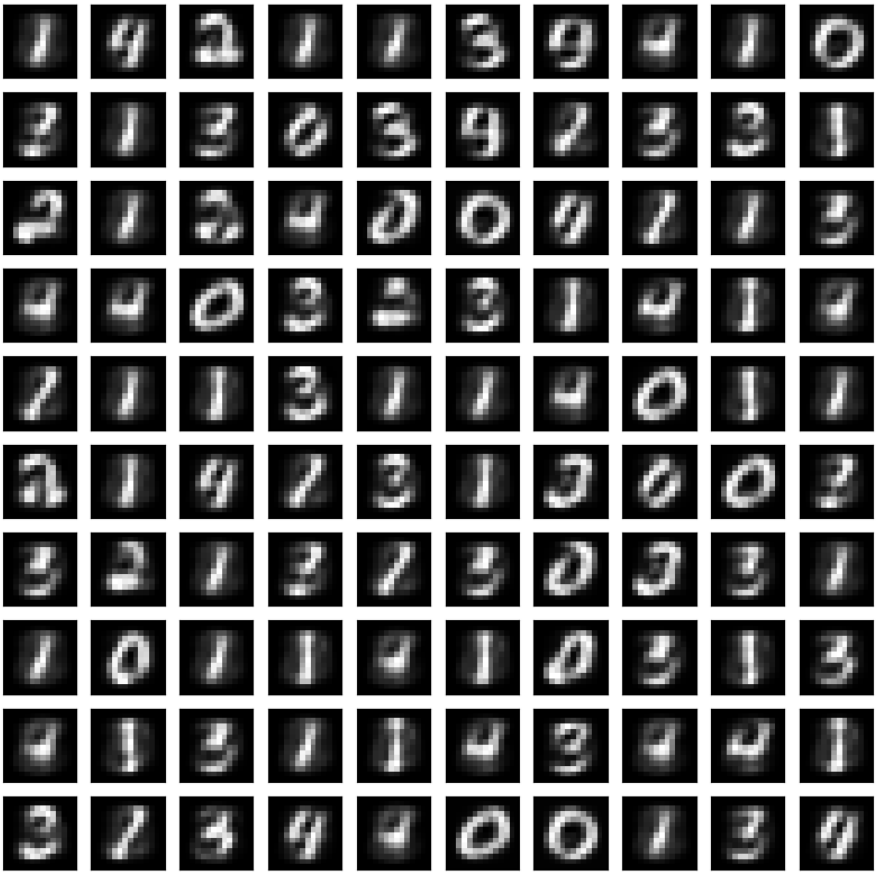


Figure 3.17: $K = 100$ images of size 14×14 learned with $M = 100, T = 10000$ initialized by $m_h^{(0)} = \text{rand(M)}$

Instead, the following results were obtained in the Hopfield case:

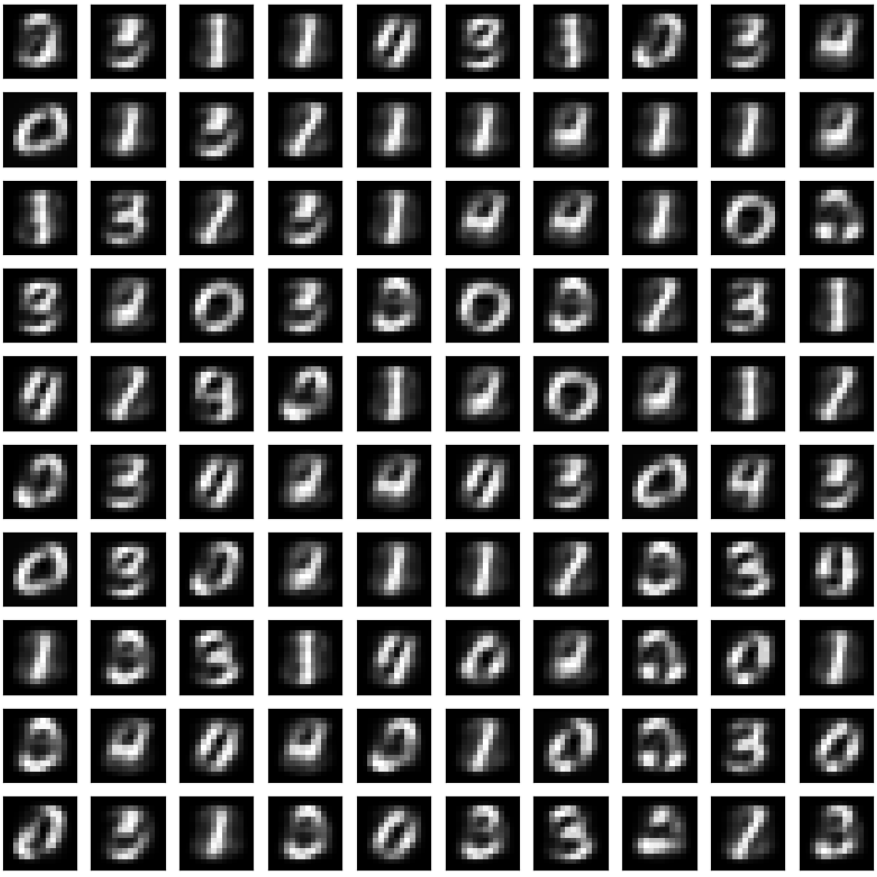
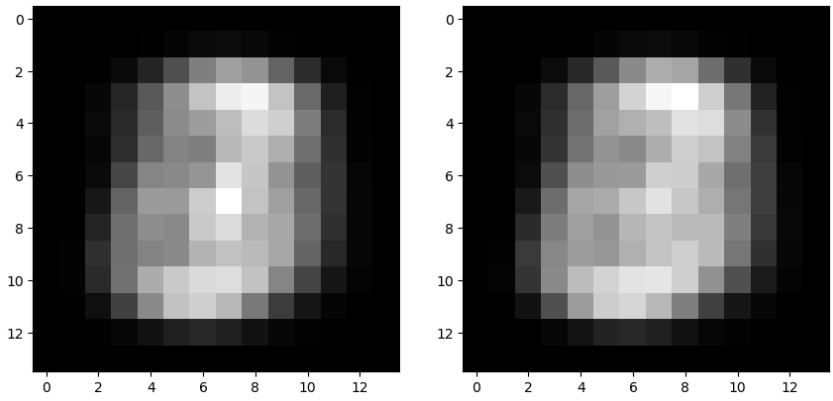


Figure 3.18: $K = 100$ images of size 14×14 learned with $M = 100, T = 10000$ initialized by $m_h^{(0)} = \text{zeros(M)}$

Subsequently, applying Gaussian mixture techniques once more yields:



(a) $m_h^{(t=0)} = \text{rand(M)}$ (b) $m_h^{(t=0)} = \text{zeros(M)}$

Figure 3.19: Gaussian mixtures of K=100 independent copies of size 14×14 learned with $M = 100, T = 10000$ in the case of $m_h^{(t=0)} = \text{zeros(M)}$ (a) and $m_h^{(t=0)} = \text{rand(M)}$ (b).

Finally, by applying the two clustering steps presented in 3.4.2, we obtained the results depicted in Figs. 3.20 and 3.21 using the *rand*() initialization and considering $N_M = 6$ clusters.

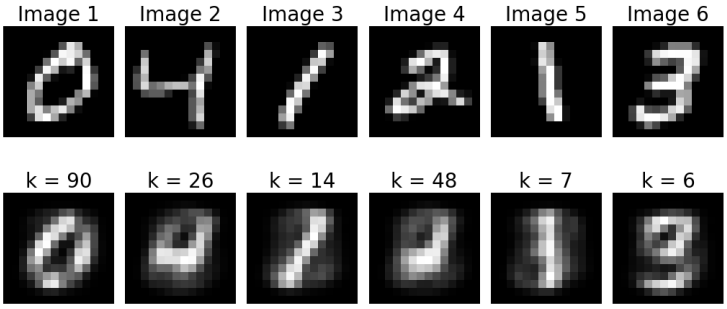


Figure 3.20: 14×14 images with their corresponding most similar EP copy found, in the case of $M = 100, K = 100, T = 10000, m_h^{(t=0)} = \text{rand(M)}$ for the digits 0,1,2,3,4.

Figure 3.21: Clustering results for 14×14 images of digits 0,1,2,3 and 4, partitioned into $N_M = 6$ clusters with $M = 100, K = 100, T = 10000, m_h^{(t=0)} = \text{rand(M)}$.

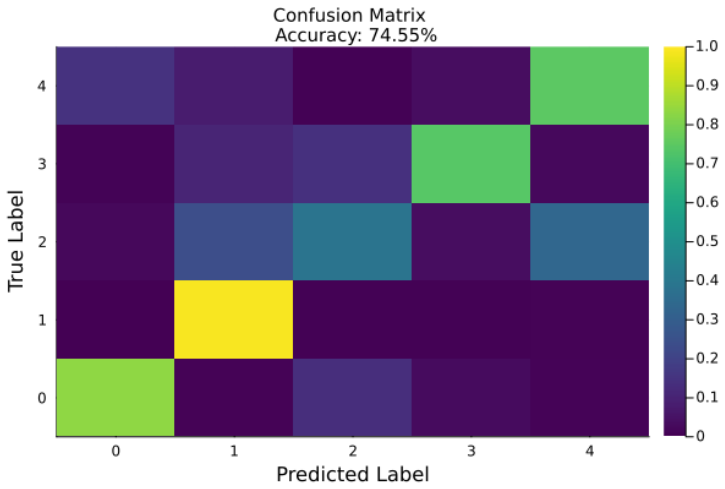


Figure 3.22: Confusion matrix for 6 clusters, based on 14×14 images of digits 0,1,2,3 and 4, with $M = 100, K = 100, T = 10000, m_h^{(t=0)} = \text{rand(M)}$.

Despite achieving a high Pearson correlation coefficient of approximately 99% during training, our clustering method does not work perfectly, with an overall accuracy of 74.55%. In particular, the algorithm struggles to correctly cluster the digit 2 which is frequently misclassified into clusters associated with other digits, as illustrated by the confusion matrix in Fig. 3.22. Moreover, different values of N_M were tested but the best outcomes were observed when $N_M = 6$. This might seem unusual since we are considering only 5 digits. However, by looking at the orange and the yellow clusters in Fig. 3.21, it is evident that the clustering algorithm better distinguishes between thick and thin 0s compared to other digits.

Chapter 4

Integration of Population Dynamics in RBM Training

In this final chapter, we will present the results achieved by integrating a population dynamics scheme into the training process of an RBM.

Population dynamics is a mathematical framework originally introduced in the context of mean field equations by Abou-Chacra et al. in 1973 [24]. This formalism was further developed by Mezard and Parisi in 2001 [25] within the spin glass theory. For a detailed explanation, refer to [26], which also inspired the development of our methodology.

4.1 Introducing Unequal Copy Probabilities

In our previous training, we assumed that each copy was equally probable, with a probability $\pi_k = \frac{1}{K}\ \forall k$ of occurring. Now, we will relax this assumption of equal probability.

Let us consider again our full distribution given by a Gaussian mixture of all K independent Gaussian distributions:

$$Q(\mathbf{v}) = \sum_{k=1}^{K} \pi_k \, \mathcal{N}^{(k)}(\mathbf{v}|\mathbf{d}^{(k)}, \mathbf{C_v}^{(k)}) \tag{4.1}$$

We will now express the probability of each copy in terms of entropy. In information theory, entropy quantifies the unpredictability or information content associated with outcomes drawn from a distribution. High-probability events have low entropy, while low-probability events have high entropy.

For a continuous random variable x with probability density $P(x)$, the entropy is defined as: $S(x) = -\int P(x) \log P(x) dx$. In the specific case of a multivariate Gaussian distribution $\mathcal{N}(\mathbf{x}; \mu, \Sigma)$, where $\mathbf{x}$ is a D-dimensional vector, the entropy is given by:

$$S = \frac{1}{2} \log(|\mathbf{\Sigma}|) + \frac{D}{2}(1 + \log(2\pi)) \tag{4.2}$$

Here, the negative sign indicates that high probability events have less entropy. For our purposes, we can neglect the second term, which is constant, and focus only on the relative entropy values. In the context of Eq. 4.1, the entropy associated with the k-th copy is: $S^{(k)} = \frac{1}{2} \log(|\mathbf{C_v^{(k)}}|)$. Afterward, we can introduce a modified version of the entropy $\tilde{S}^{(k)} = \frac{1}{2} \log(\det(\mathbf{C_v^{(k)}})) - \max_{k'} S^{(k')}$ which will enter in the expression of the probabilities (Eq. 4.3) and will enhance numerical stability. The probability of each copy is then written as a normalized exponential function, also known as the softmax function:

$$\pi_k = \frac{e^{\tilde{S}^{(k)}}}{\sum_{k'} e^{\tilde{S}^{(k')}}} \tag{4.3}$$

4.2 Results Obtained

4.2.1 Method 1

In the previous simulations, we had K distributions that evolved independently of one another and contributed with the same weight to the full probability distribution. Now, we introduce an auxiliary Gaussian distribution and a temporary one that will allow us to model the interactions between the copies. The details of this implementation are provided in the Appendix (see Alg. 1).

As shown in Fig. 4.1, after approximately $t \approx 100$ epochs, the gradient begins to oscillate significantly.

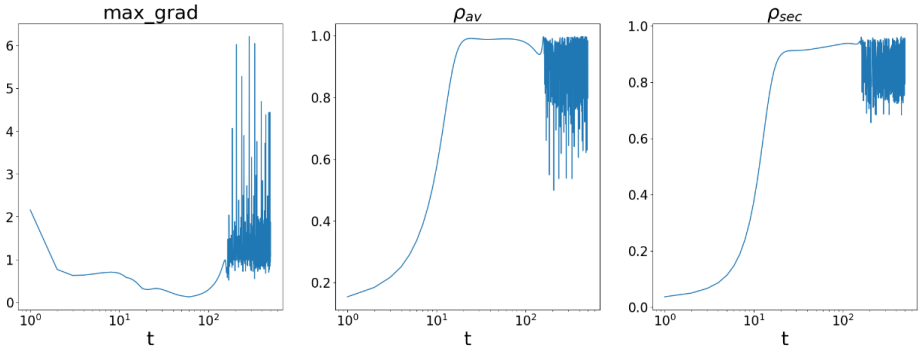


Figure 4.1: Variation of maximum gradient, ρ_{av} and ρ_{sec} for images of size 14×14 with $M = 10, K = 100, m_h^{(t=0)} =$ rand(M) obtained using a population dynamics scheme (first method).

For high correlations ($\rho \simeq 1$) and vanishing maximum gradient, all copies converge to the same distribution, given by the superposition of the 0 and 1 digits (as illustrated in Fig. 4.2). Conversely, when the maximum gradient explodes and correlations decrease, all the copies at different epochs converge to a single digit type: either 0 or 1 (see Fig. 4.3).

Figure 4.2: 5x5 grid showing 25 copies learned applying a population dynamics scheme (first method) for images of size 14×14 with $M = 20, K = 100$, at $t = 180$.

Figure 4.3: 5x5 grid showing 25 copies learned applying a population dynamics scheme (first method) for images of size 14×14 with $M = 20, K = 100$, at $t = 200$.

4.2.2 Method 2

In this second approach, we integrate the "classical" method, where all the copies evolve independently and contribute to the full distribution according to Eq. 4.3, with the population dynamics scheme discussed in the previous section. Detailed steps of the algorithm are provided in the Appendix (see Alg. 2). Initially, we perform EP_step_max iterations before updating the gradient. However, unlike the previous method, we introduce a population dynamics interaction at EP_step_/2 iterations, as described earlier. This approach allows for some independent "classical" iterations before estimating the gradient, while still enabling interaction among the different copies. The aim is to achieve a balance between independent training and collaborative interaction among the copies.

As illustrated in Fig. 4.4, the maximum gradient, ρ_{av} and ρ_{sec} obtained exhibit a similar behaviour to those from the previous method (see Fig. 4.1). However, unlike before, the copies do not converge anymore to the same distribution, as shown in Fig. 4.5.

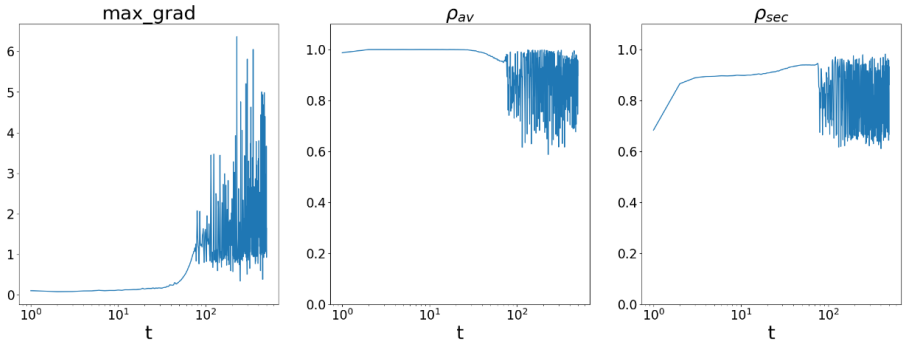


Figure 4.4: Variation of maximum gradient, ρ_{av} and ρ_{sec} for images of size 14×14 with $M = 10, K = 100, m_h^{(t=0)} = \text{rand(M)}$ obtained using a population dynamics scheme (second method).

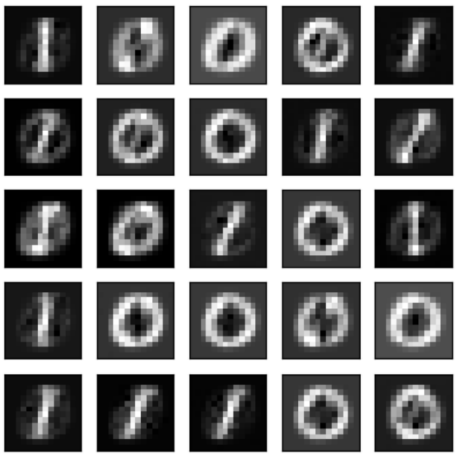


Figure 4.5: 5x5 grid showing 25 copies learned applying a population dynamics scheme (second method) for images of size 14×14 with $M = 20, K = 100$, at $t = 500$.

Chapter 5

Conclusion

In this thesis, we explored a new training method for RBMs using the EP algorithm to approximate the moments of the model distribution involved in likelihood maximization. Our main goal was to test the accuracy and performance of this model on the MNIST dataset, with a specific focus on the digits 0 and 1.

We began by discussing the theory behind RBMs and EP, followed by a detailed presentation of the mathematical framework necessary to apply EP to a Gaussian RBM. This Gaussian variant of RBM assumed a Gaussian prior distribution for the hidden units and a two-component Gaussian mixture distribution for the visible ones.

Initially, we obtained unsatisfactory results in the case of a single copy ($K = 1$), evidenced by low correlation values even after $T = 10000$ epochs. The main reason for this outcome was the model converging to single different digit types (0 or 1) at various epochs. For this reason we tested the model with multiple independent copies ($K = 100$), leveraging a Gaussian mixture technique to incorporate the contribution of each copy. This approach significantly improved digit recognition accuracy, demonstrating the method's effectiveness.

Next, we examined the model's performance for different values of M (number of hidden units) and K. It was observed that even with relatively small val-

ues of K and M, the model performed adequately. Increasing K and M yielded slightly better results, although this has also a computational cost to take into account. We then selected the regimes (at convergence) ensuring efficient training without excessive computational costs and conducted some clustering tests on the dataset.

Subsequently, we extended our experiments to include digits from 0 to 4. This added complexity allowed us to test the EP algorithm in more challenging scenarios. The results were promising; however, the learned digits did not perfectly replicate the original ones, thus negatively affecting the accuracy of the clustering .

Lastly, we explored an alternative approach by integrating a population dynamics scheme into training, facilitating enhanced exploration of solution space and achieving more efficient and accurate training.

5.1 Future developments

Clearly, there remains a significant amount of research to be conducted in this area. Future studies could expand my thesis work by including broader sets of digits or focusing on different digit pairs beyond 0 and 1. Investigating how well the model performs with different digits could better uncover the strengths and limitations of this approach. Moreover, the integration of more sophisticated population dynamics schemes into the training process presents an exciting opportunity for advancement in the field. Finally, although MNIST is widely used as a benchmark, training RBMs using EP on different datasets with greater dimensionality, variability, or noise levels could provide a more realistic assessment of its robustness and scalability.

Appendix

Algorithm 1: Population Dynamics scheme 1 implemented

for $t \in 1:T$ **do**
  for *EP_step* $\in$ *1:EP_step_max* **do**
    for $k \in 1:K$ **do**
      // Draw N random indices uniformly from 1 to K:
      idx = rand(1:K, N)
      // Copy selected values from current parameters ($\mathbf{a\_cur}, \mathbf{b\_cur}$) to auxiliary ones ($\mathbf{a\_aux}, \mathbf{b\_aux}$):
      for $i \in 1:N$ **do**
        a_aux[i] = a_cur[i, idx[i]] ; b_aux[i] = b_cur[i, idx[i]]
      // Compute temporary parameters (**a_temp**,**b_temp**) using (**a_aux**, **b_aux**) as arguments for the EP algorithm;
      // Compute covariance matrix $\mathbf{C_v}$ using **a_temp**, **b_temp**;
      // Compute entropy S[k] according to:
      $S[k] = 0.5\log(\det(C_v))$
    // Calculate probabilities $\pi[k]$ associated to the k-th copy using entropy S[k];
    // Draw K indices k_{temp} according to the distribution π;
    // Copy selected values from temporary parameters (**a_temp**, **b_temp**) to current ones (**a_cur**, **b_cur**);
  // Compute $\langle v_i \rangle_{EP}$ and $\langle v_i v_j \rangle_{EP}$, using **a_cur** and **b_cur**;
  // Calculate $\boldsymbol{\pi}$ using the updated **a_cur** and **b_cur**;
  // Compute Gaussian mixture using the updated π;
  // Compute the gradient and update the weights W;

Algorithm 2: Population Dynamics scheme 2 implemented

for $t \in 1:T$ **do**

  // Population dynamic interaction after half of EP_step_max steps

  if *EP_step==div(EP_step_max, 2)* **then**

    for *EP_step* $\in$ *1:EP_step_max* **do**

      for $k \in 1:K$ **do**

        // Draw N random indices uniformly from 1 to K:

        idx = rand(1:K, N)

        // Copy selected values from current parameters ($\mathbf{a\_cur}, \mathbf{b\_cur}$) to auxiliary ones ($\mathbf{a\_aux}, \mathbf{b\_aux}$):

        for $i \in 1:N$ **do**

          a_aux[i] = a_cur[i, idx[i]] ; b_aux[i] = b_cur[i, idx[i]]

        // Compute temporary parameters ($\mathbf{a\_temp}, \mathbf{b\_temp}$) using ($\mathbf{a\_aux}$, $\mathbf{b\_aux}$) as arguments for the EP algorithm;

        // Compute covariance matrix $\mathbf{C_v}$ using **a_temp**, **b_temp**;

        // Compute entropy S[k] according to:

        $S[k] = 0.5\log(\det(C_v))$

      // Calculate probabilities $\pi[k]$ associated to the k-th copy using entropy S[k];

      // Draw K indices k_{temp} according to the distribution π;

      // Copy selected values from temporary parameters ($\mathbf{a\_temp}$, $\mathbf{b\_temp}$) to current ones ($\mathbf{a\_cur}$, $\mathbf{b\_cur}$);

    else

      // Normal run of the EP algorithm

  // Compute $\langle v_i \rangle_{EP}$ and $\langle v_i v_j \rangle_{EP}$, using **a_cur** and **b_cur**;

  // Calculate $\boldsymbol{\pi}$ using the updated **a_cur** and **b_cur**;

  // Compute Gaussian mixture using the updated π;

  // Compute the gradient and update the weights W;

Bibliography

[1] E. Ising, "Beitrag zur theorie des ferromagnetismus," *Zeitschrift für Physik*, vol. 31, pp. 253–258, 1925.

[2] S. Amari, "Learning patterns and pattern sequences by self-organizing nets of threshold elements," *IEEE Transactions on Computers*, vol. C-21, pp. 1197–1206, 1972.

[3] J. J. Hopfield, "Neural networks and physical systems with emergent collective computational abilities.," *Proceedings of the National Academy of Sciences of the United States of America*, vol. 79, pp. 2554–2558, Apr. 1982.

[4] G. E. Hinton and J. Sejnowski, "Optimal perceptual inference," 1983.

[5] P. Smolensky, "Information processing in dynamical systems: Foundations of harmony theory," *Parallel Distributed Process*, vol. 1, 01 1986.

[6] G. E. Hinton, "Training products of experts by minimizing contrastive divergence," *Neural Comput.*, vol. 14, p. 1771–1800, aug 2002.

[7] G. E. Hinton, S. Osindero, and Y. W. Teh, "A fast learning algorithm for deep belief nets," *Neural Computation*, vol. 18, pp. 1527–1554, 2006.

[8] Y. W. Teh and G. E. Hinton, "Rate-coded restricted boltzmann machines for face recognition," in *Advances in Neural Information Processing Systems*, vol. 13, pp. 908–914, 2001.

[9] V. Nair and G. E. Hinton, "Rectified linear units improve restricted boltzmann machines," in *Proceedings of the 27th international conference on machine learning (ICML-10)*, pp. 807–814, 2010.

[10] D. Koller and N. Friedman, *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[11] A. Fischer and C. Igel, "Training restricted boltzmann machines: An introduction," *Pattern Recognition*, vol. 47, no. 1, pp. 25–39, 2014.

[12] T. Tieleman, "Training restricted boltzmann machines using approximations to the likelihood gradient," in *Proceedings of the 25th international conference on Machine learning*, pp. 1064–1071, 2008.

[13] G. Desjardins, A. C. Courville, Y. Bengio, P. Vincent, and O. Delalleau, "Parallel tempering for training of restricted boltzmann machines," 2010.

[14] M. Opper and O. Winther, "Gaussian processes for classification: Mean-field algorithms," *Neural computation*, vol. 12, no. 11, pp. 2655–2684, 2000.

[15] M. Opper and O. Winther, "Adaptive and self-averaging thouless-anderson-palmer mean-field theory for probabilistic modeling," *Physical Review E*, vol. 64, no. 5, p. 056131, 2001.

[16] T. P. Minka and R. Picard, *A family of algorithms for approximate bayesian inference*. PhD thesis, USA, 2001.

[17] M. Opper and O. Winther, "A bayesian approach to on-line learning," in *On-line learning in neural networks*, pp. 363–378, Cambridge University Press, 1999.

[18] K. Murphy, Y. Weiss, and M. I. Jordan, "Loopy belief propagation for approximate inference: An empirical study," *arXiv preprint arXiv:1301.6725*, 2013.

[19] S. Wang, "Expectation propagation algorithm," 2011.

[20] S. Lauritzen, "Propagation of probabilities, means and variances in mixed graphical association models," *Journal of the American Statistical Association*, vol. 87, pp. 1098–1108, 1992.

[21] X. Boyen and D. Koller, "Tractable inference for complex stochastic processes," 1998.

[22] P. S. Maybeck, *Stochastic models, estimation, and control*. Academic press, 1982.

[23] Y. LeCun, C. Cortes, and C. J. Burges, "The mnist database of handwritten digits." Website. `http://yann.lecun.com/exdb/mnist/`.

[24] R. Abou-Chacra, D. Thouless, and P. Anderson, "A selfconsistent theory of localization," *Journal of Physics C: Solid State Physics*, vol. 6, no. 10, p. 1734, 1973.

[25] M. Mézard and G. Parisi, "The bethe lattice spin glass revisited," *The European Physical Journal B-Condensed Matter and Complex Systems*, vol. 20, pp. 217–233, 2001.

[26] M. Mezard and A. Montanari, *Information, physics, and computation.* Oxford University Press, 2009.